Functions in ORACLE-SQL

ORACLE

Topics: #sql #functions #aggregation #formatting #datetime #timestamps #strings #numbers #data-types #math #data-cleaning #query

My obsidian notes.

FUNCTIONS

Functions can be categorized as follows.

1. Single row functions
2. Group functions

SINGLE ROW FUNCTIONS

Single row functions fall into five categories, and these are applied individually to each row, producing distinct outputs for each row.

1. Numeric functions
2. String functions
3. Date functions
4. Miscellaneous functions
5. Conversion functions

BEFORE WE GET STARTED

In almost all the commands below you will find a keyword `dual`.

What is `DUAL` ?

In Oracle, the `DUAL` table is a one-row, one-column table present by default in all Oracle databases. It is often used for testing and storing the results of expressions that involve computations or function calls. The table has a single column named `"DUMMY"` with a data type of `VARCHAR2(1)` and a single row containing the value `'X'`.

What can I do with it?

For example, you might use the `DUAL` table when testing a function or expression like this:

```
SQL> SELECT SYSDATE FROM DUAL;

SYSDATE
---------
14-NOV-21
```

This query returns the current date and time, and the `DUAL` table is used as a dummy table to ensure the select statement is syntactically correct. As for other usages you are about to find out in the following sections.

Why does it exist?

While the `DUAL` table itself doesn't serve a complex purpose, it provides a convenient way to perform operations that require a table when no other table is applicable.

USING - `NUMERIC FUNCTIONS`

Abs	**Sign**	**Sqrt**	**Mod**	**Nvl**
Power	**Exp**	**Ln**	**Log**	**Ceil**
Floor	**Round**	**Trunk**	**Bitand**	**Greatest**
Least	**Coalesce**			

USING - `ABS`

(Absolute value is the magnitude of a real number regardless of its sign.)

Syntax

```
abs(_value_)
```

Example

```
SQL> select abs(5), abs(-5), abs(0), abs(null) from dual;

 ABS(5) ABS(-5)   ABS(0)  ABS(NULL)
 ------ -------   ------- ---------
 5       5          0
```

USING - `SIGN`

Yields an output that returns the sign of a value.

Syntax

```
sign(_value_)
```

Example

```
SQL> select sign(5), sign(-5), sign(0), sign(null) from dual;

 SIGN(5)    SIGN(-5)    SIGN(0)  SIGN(NULL)
---------- ---------- ---------- --------------
 1          -1           0
```

USING - `SQRT`

Returns the square root ($\sqrt{n}$) of a value.

Syntax

```
sqrt(_value_) -- here value must be positive
```

Example

```
SQL> select sqrt(4), sqrt(0), sqrt(null), sqrt(1) from dual;

 SQRT(4)     SQRT(0)     SQRT(NULL)     SQRT(1)
---------- ---------- --------------- ----------
    2           0                          1
```

USING - `MOD`

Returns the remainder.

Syntax

```
mod (_value, divisor_)
```

Example

```
SQL> select mod(7,4), mod(1,5), mod(null,null), mod(0,0), mod(-

7,4) from dual;

 MOD(7,4)    MOD(1,5)    MOD(NULL,NULL)      MOD(0,0)    MOD(-7,4)
------------ ---------- --------------------- ----------- -------------
    3           1                                0            -3
```

USING - `NVL`

Fills the `null` values with specified value

Syntax

```
nvl(_null_col, replacement_value_)
```

Example

```
 SQL> select * from student; -- here for 3rd row marks value is null

     NO   NAME   MARKS
     --- ------- ---------
     1    a       100
     2    b       200
     3    c
```

```
SQL> select no, name, nvl(marks,300) from student;

    NO  NAME    NVL(MARKS,300)
    --- ------- --------------------
    1   a        100

    2   b        200

    3   c        300
```

```
SQL> select nvl(1,2), nvl(2,3), nvl(4,3), nvl(5,4) from dual;

 NVL(1,2)   NVL(2,3)   NVL(4,3)   NVL(5,4)
---------- ---------- ---------- ----------
        1          2          4          5
```

```
SQL> select nvl(0,0), nvl(1,1), nvl(null,null), nvl(4,4)
from dual;

NVL(0,0)    NVL(1,1)    NVL(null,null)   NVL(4,4)
---------- ---------- ----------------- ----------
     0          1                            4
```

USING - `POWER`

Raises a value to a given exponent (x^n)

Syntax

```
power (_value, exponent_)
```

Example

```
SQL> select power(2,5), power(0,0), power(1,1), power(null,null), power(2,-5) from dual;

POWER(2,5) POWER(0,0) POWER(1,1) POWER(NULL,NULL) POWER(2,-5)
---------- ---------- ---------- ---------------- -----------
        32          1          1                       .03125
```

USING - `EXP`

Basically performs - e^n

Syntax

```
exp(_value_)
```

Example

```
SQL> SELECT exp(1), exp(2), exp(0), exp(null), exp(-2) FROM dual;

    EXP(1)     EXP(2)     EXP(0)  EXP(NULL)    EXP(-2)
---------- ---------- ---------- ---------- ----------
2.71828183  7.3890561          1            .135335283
```

USING - `LN`

`LN` stands for the natural logarithm. The natural logarithm is the logarithm to the base e where e is Euler's number, an irrational constant approximately equal to 2.71828.

Syntax

```
ln(_value_) -- here value must be greater than zero which is positive only.
```

Example

```
SQL> SELECT ln(1), ln(2), ln(null) FROM dual;

LN(1)        LN(2)       LN(NULL)
------------ ----------- ------------
0            .693147181
```

`LN` and `EXP` are reciprocal to each other.

```
EXP(3) = 20.0855369
LN(20.0855369) = 3
```

USING - `LOG`

Basically $log_{10}(n)$

Syntax

```
log (10, _value_) -- here value must be greater than zero which is positive only.
```

Example

```
SQL> select log(10,100), log(10,2), log(10,1), log(10,null) from dual;

LOG(10,100)  LOG(10,2)  LOG(10,1) LOG(10,NULL)
----------- ---------- ---------- ------------
          2 .301029996          0
```

```
LN (value) = LOG (EXP(1), value)
```

```
SQL> select ln(3), log(exp(1),3) from dual;
```

```
     LN(3) LOG(EXP(1),3)
---------- -------------
1.09861229    1.09861229
```

USING - `CEIL`

Produces a whole number that is greater than or equal to the specified value

Syntax

```
ceil (_value_)
```

Example

```
SQL> select ceil(5), ceil(5.1), ceil(-5), ceil( -5.1), ceil(0), ceil(null) from dual;

   CEIL(5)  CEIL(5.1)   CEIL(-5) CEIL(-5.1)    CEIL(0) CEIL(NULL)
---------- ---------- ---------- ---------- ---------- ----------
         5          6         -5         -5          0
```

USING - `FLOOR`

Produces a whole number that is less than or equal to the specified value.)
Syntax

```
floor (_value_)
```

Example

```
SQL> select floor(5), floor(5.1), floor(-5), floor( -5.1), floor(0), floor(null) from dual;

  FLOOR(5) FLOOR(5.1)  FLOOR(-5) FLOOR(-5.1)   FLOOR(0) FLOOR(NULL)
---------- ---------- ---------- ----------- ---------- -----------
         5          5         -5          -6          0
```

USING - `ROUND`

---

rounds numbers to a given number of digits of precision

Syntax

```
round (_value, precision_)
```

Example

```
SQL> select round(123.2345), round(123.2345,2), round(123.2354,2) from dual;

ROUND(123.2345) ROUND(123.2345,2) ROUND(123.2354,2)
--------------- ----------------- -----------------
            123            123.23            123.24
```

```
SQL> select round(123.2345,-1), round(123.2345,-2), round(123.2345,-3), round(123.2345,-4) from dual;

ROUND(123.2345,-1) ROUND(123.2345,-2) ROUND(123.2345,-3) ROUND(123.2345,-4)
------------------ ------------------ ------------------ ------------------
               120                100                  0                  0
```

```
SQL> select round(123,0), round(123,1), round(123,2) from dual;

ROUND(123,0) ROUND(123,1) ROUND(123,2)
------------ ------------ ------------
         123          123          123
```

```
SQL> select round(-123,0), round(-123,1), round(-123,2) from dual;

ROUND(-123,0) ROUND(-123,1) ROUND(-123,2)
------------- ------------- -------------
         -123          -123          -123
```

```
SQL> select round(123,-1), round(123,-2), round(123,-3), round(-123,-1), round(-123,-2), round(-123,-3) from dual;

ROUND(123,-1) ROUND(123,-2) ROUND(123,-3) ROUND(-123,-1) ROUND(-123,-2) ROUND(-123,-3)
------------- ------------- ------------- -------------- -------------- --------------
          120           100             0           -120           -100              0
```

```

SQL> select round(null,null), round(0,0), round(1,1), round(-1,-1), round(-2,-2) from dual;

ROUND(NULL,NULL) ROUND(0,0) ROUND(1,1) ROUND(-1,-1) ROUND(-2,-2)
---------------- ---------- ---------- ------------ ------------
                          0          1            0            0

```

USING - `TRUNC`

Truncates or shaves off digits of precision from a number

Syntax

```
trunc (_value, precision_)
```

Example

```
SQL> select trunc(123.2345), trunc(123.2345,2), trunc(123.2354,2) from dual;

TRUNC(123.2345) TRUNC(123.2345,2) TRUNC(123.2354,2)
```

```
--------------- ----------------- -----------------
            123            123.23            123.23
```

```
SQL> select trunc(123.2345,-1), trunc(123.2345,-2), trunc(123.2345,-3), trunc(123.2345,-4) from dual;

TRUNC(123.2345,-1) TRUNC(123.2345,-2) TRUNC(123.2345,-3) TRUNC(123.2345,-4)
------------------ ------------------ ------------------ ------------------
               120                100                  0                  0
```

```
SQL> select trunc(123,0), trunc(123,1), trunc(123,2) from dual;

TRUNC(123,0) TRUNC(123,1) TRUNC(123,2)
------------ ------------ ------------
         123          123          123
```

```
SQL> select trunc(-123,0), trunc(-123,1), trunc(-123,2) from dual;


TRUNC(-123,0) TRUNC(-123,1) TRUNC(-123,2)
------------- ------------- -------------
         -123          -123          -123
```

```
SQL> SELECT
  2    TRUNC(123, -1),
  3    TRUNC(123, -2),
  4    TRUNC(123, -3),
  5    TRUNC(-123, -1),
  6    TRUNC(-123, 2),
  7    TRUNC(-123, -3)
  8  FROM dual;

TRUNC(123,-1) TRUNC(123,-2) TRUNC(123,-3) TRUNC(-123,-1) TRUNC(-123,2) TRUNC(-123,-3)
------------- ------------- ------------- -------------- ------------- --------------
          120           100             0           -120          -123              0
```

```
SQL> select trunc(null,null), trunc(0,0), trunc(1,1), trunc(-1,-1), trunc(-2,-2) from dual;

TRUNC(NULL,NULL) TRUNC(0,0) TRUNC(1,1) TRUNC(-1,-1) TRUNC(-2,-2)
---------------- ---------- ---------- ------------ ------------
                          0          1            0            0
```

USING - `BITAND`

Performs bitwise and operation.

Syntax

```
bitand (_value1, value2_)
```

Example

```
SQL> select bitand(2,3), bitand(0,0), bitand(1,1), bitand(null,null), bitand(-2,-3) from dual;

```

```
BITAND(2,3) BITAND(0,0) BITAND(1,1) BITAND(NULL,NULL) BITAND(-2,-3)
----------- ----------- ----------- ----------------- -------------
          2           0           1                              -4
```

USING - `GREATEST`

Returns the greatest number.

Syntax

```
greatest (_value1, value2, value3 … valuen_)
```

Example

```
SQL> select greatest(1, 2, 3), greatest(-1, -2, -3) from dual;

GREATEST(1,2,3) GREATEST(-1,-2,-3)
--------------- ------------------
              3                 -1
```

Additional Notes

1. In the event that all values within the dataset are zeros, the system will render a result of zero.
2. In the circumstance where all parameters are null, the system will yield no output.
3. Should any of the parameters be identified as null, the system will produce no output.

USING - `LEAST`

Returns the least/lowest number.

Syntax

```
least (_value1, value2, value3 … valuen_)
```

Example

```
SQL> select least(1, 2, 3), least(-1, -2, -3) from dual;

LEAST(1,2,3) LEAST(-1,-2,-3)
------------ ---------------
           1              -3
```

Additional Notes

1. In the event that all values within the dataset are zeros, the system will render a result of zero.
2. In the circumstance where all parameters are null, the system will yield no output.

3. Should any of the parameters be identified as null, the system will produce no output.

USING - `COALESCE`

Returns first `non-null` value.

Syntax

```
coalesce (_value1, value2, value3 … valuen_)
```

Example

```
SQL> select coalesce(1,2,3), coalesce(null,2,null,5) from dual;

COALESCE(1,2,3) COALESCE(NULL,2,NULL,5)
--------------- -----------------------
              1                       2
```

USING - `STRING FUNCTIONS`

Initcap	**Upper**	**Lower**	**Length**
Rpad	**Lpad**	**Ltrim**	**Rtrim**
Trim	**Translate**	**Replace**	**Soundex**
Concat('II Concatenation Operation')	**Ascii**	**Chr**	
Substr	**Instr**	**Decode**	**Greatest**
Least	**Coalesce**		

USING - `INITCAP`

Capitalizes the initial letter of every word in a string.

Syntax

```
initcap(_string_)
```

Example

```
SQL> select initcap('computer') from dual;

INITCAP(
--------
Computer

SQL> select initcap('computer science') from dual;
```

```
INITCAP('COMPUTE
----------------
Computer Science
```

USING - `UPPER`

Converts the given string to uppercase.

Syntax

```
upper(_string_)
```

Example

```
SQL> select upper('computer') from dual;

UPPER('C
--------
COMPUTER
```

USING - `LOWER`

Converts the given string to lowercase.

Syntax

```
lower(_string_)
```

Example

```
SQL> select lower('COMPUTER') from dual;

LOWER
-----------
computer
```

USING - `LENGTH`

Returns the length of a given string.

Syntax

```
length (_string_)
```

Example

```
SQL> select length('computer') from dual;

LENGTH
-----------
 8
```

USING - `RPAD`

Allows padding the right side of a column with any set of characters

Syntax

```
rpad (_string, length [, padding_char]_)
```

```
SQL> select rpad('computer',15,'*'), rpad('computer',15,'*#') from dual;
```

```
RPAD('COMPUTER' RPAD('COMPUTER'
--------------- ---------------
computer******* computer*#*#*#*
```

***Note:** The default padding character is blank space.

USING - `LPAD`

Allows padding the left side of a column with any set of characters.

Syntax

```
lpad (_string, length [, padding_char]_)
```

Example

```
SQL> select lpad('computer',15,'*'), lpad('computer',15,'*#') from dual;
```

```
LPAD('COMPUTER' LPAD('COMPUTER'
--------------- ---------------
*******computer *#*#*#*computer
```

USING - `LTRIM`

Trims specified characters from the left end of a string

Syntax

```
ltrim(_<string>_, _<chars_to_trim>_)
```

Example

```
SQL> SELECT ltrim('computer', 'co'), ltrim('computer', 'com') FROM dual;

LTRIM(       LTRIM
------------ ---------
mputer       puter
```

***Note:** Specifying the characters to be trimmed is crucial for the operation. They have to match characters to the left, starting from the first or else it doesn't change the string.

```
SQL> SELECT ltrim('computer', 'puter'), ltrim('computer', 'omputer') FROM dual;
```

```
LTRIM('C    LTRIM('C
----------  ----------
computer     computer
```

USING - `RTRIM`

Trims specified characters from the right end of a given string.

Syntax

```
rtrim(_<string>_, _<chars_to_trim>_)
```

Example

```
SQL>  select rtrim('computer','er'), rtrim('computer','ter') from dual;

RTRIM( RTRIM
------ -----
comput compu
```

```
SQL> SELECT RTRIM('computer', 'comput'), RTRIM('computer', 'compute') FROM DUAL;
```

```
RTRIM('C RTRIM('C
-------- --------
computer computer
```

USING - `TRIM`

Trims specified characters from the both sides of a string.

Syntax

```
trim(_<string>_, _<chars_to_trim>_)
```

Example

```
SQL> select trim( 'i' from 'indiani') from dual;

TRIM(
-----
ndian
```

```
SQL> select trim( leading'i' from 'indiani') from dual; -- this works as LTRIM
TRIM(L
------
ndiani
```

```
SQL> select trim( trailing'i' from 'indiani') from dual;

TRIM(T
------
indian
```

USING - `TRANSLATE`

In Oracle, the `TRANSLATE` function is used to replace each character in a string with a corresponding character at the same position in another specified string or to remove characters from the input string. The function takes three string arguments:

1. **source_string:** This is the original string where characters will be replaced or removed.
2. **search_string:** The characters in this string will be searched in the source_string for replacement or removal.
3. **replace_string:** The corresponding characters in this string will replace the characters found in the search_string.

Syntax

```
TRANSLATE(source_string, search_string, replace_string)
```

Example

```
SQL> SELECT TRANSLATE('Hello', 'el', 'XY') FROM DUAL;

TRANS
-----
HXYYo
```

In this example, the function replaces 'e' with 'X' and 'l' with 'Y', resulting in the output 'HYXXo'. Basically, every 'e' in the string maps to 'X' and every 'l' in the string maps to 'Y'.

- *It's important to note that the lengths of the `search_string` and `replace_string` must be the same, or else Oracle will raise an error. Additionally, if a character in the `source_string` is not found in the `search_string`, it remains unchanged in the output.*

USING - `REPLACE`

In Oracle, the `REPLACE` function is used to replace occurrences of a specified substring with another substring within a given string. The syntax for the `REPLACE` function is as follows:

```
REPLACE(original_string, search_string, replace_string)
```

- `original_string`: This is the string in which you want to replace occurrences of a substring.
- `search_string`: This is the substring you want to find and replace.
- `replace_string`: This is the substring that will replace each occurrence of the `search_string` in the `original_string`.

Example:

```
SELECT REPLACE('apple,orange,banana', ',', '|') FROM DUAL;
```

```
REPLACE('APPLE,ORAN
-------------------
apple|orange|banana
```

- *In this example, the function replaces all occurrences of ',' (comma) with '|' (pipe), resulting in the output 'apple|orange|banana'.*
- *It's important to note that the `REPLACE` function is case-sensitive. If you need a case-insensitive replacement, you can use the `REGEXP_REPLACE` function with appropriate regular expression options.**

```
SELECT REGEXP_REPLACE('apple,Orange,banana', ',', '|', 1, 0, 'i') FROM DUAL;
```

```
REGEXP_REPLACE('APP
-------------------
apple|Orange|banana
```

- *In this example, the 'i' option makes the replacement case-insensitive.*

USING - `SOUNDEX`

In Oracle, the `SOUNDEX` function is used to retrieve a phonetic representation of a string based on its English pronunciation. This function assigns a four-character code to each string, and strings that have similar pronunciation often have the same or similar SOUNDEX codes.

Syntax

```
SOUNDEX(string)
```

Example

```
SQL> SELECT last_name, first_name
     FROM hr.employees
     WHERE SOUNDEX(last_name)
         = SOUNDEX('SMYTHE')
     ORDER BY last_name, first_name;

LAST_NAME  FIRST_NAME
---------- ----------
Smith      Lindsey
Smith      William
```

```
SQL> SELECT SOUNDEX('Oracle') FROM DUAL;

SOUN
----
O624
```

The output of this query will be the SOUNDEX code for the string 'Oracle'. It's important to note that the SOUNDEX function is case-insensitive, and it works best with English words.

Keep in mind that the SOUNDEX algorithm is primarily designed for English names and may not work as effectively for words from other languages. Additionally, it may not be suitable for all types of data, and its use is often limited to certain cases where phonetic similarity is more important than exact character matching.

USING - `CONCAT`

Can be used to combine two strings only

Syntax

```
concat(string1, string2)
```

Example

```
SQL> SELECT concat('computer', ' operator') FROM dual;

CONCAT('COMPUTER'
-------------------------
computer operator
```

If you want to combine more than two strings, you have to use the concatenation operator (`||`).

```
SQL> SELECT 'how' || ' are' || ' you' FROM dual;

'HOW'||'ARE
---------------
how are you
```

USING - `ASCII`

This will return the decimal representation in the database character set of the first character of the string.

Syntax

```
ascii(string)
```

Example

```
SQL> SELECT ascii('a'), ascii('apple') FROM dual;
```

```
ASCII('A') ASCII('APPLE')
------------ ------------------
97         97
```

USING - `CHR`

Produces the character having the binary equivalent to the string in either the database character set or the national character set.

Syntax

```
chr(number)
```

Example

```
SQL> SELECT chr(97) FROM dual;

CHR
-----
a
```

USING - `SUBSTR`

Can be used to extract substrings.

Syntax

```
substr(string, start_chr_count, no_of_chars)
```

Example

```
SQL> SELECT substr('computer',2), substr('computer',2,5), substr('computer',3,7) FROM dual;

SUBSTR( SUBSTR SUBSTR
------- ------ -------
omputer omput  mputer
```

Additional Notes

---

1. In the event that the `no_of_chars` parameter is provided with a negative value, the system will produce no output.
2. Should all parameters, excluding the `string`, be either null or zero, the result will be an empty display.
3. If the `no_of_chars` parameter exceeds the length of the string, the system will disregard the excess and compute the result based on the original length of the string.
4. When the `start_chr_count` parameter is assigned a negative value, the substring extraction will commence from the right end of the original string.

Here's how the indexing works:

```
1   2   3   4   5   6   7   8
C   O   M   P   U   T   E   R
-8 -7  -6  -5  -4  -3  -2  -1
```

USING - `INSTR`

Allows you to search through a string for a set of characters.

Syntax

```
instr(string, search_chars,start_chr_count, occurrence)
```

Example

```
SQL> SELECT instr('information', 'o', 4, 1), instr('information', 'o', 4, 2) FROM dual;

INSTR('INFORMATION','O',4,1) INSTR('INFORMATION','O',4,2)
---------------------------------- -------------------------------------
4                              10
```

Additional Notes

1. In the absence of specific values for the parameters `start_chr_count` and `occurrence`, the search will commence from the beginning, identifying solely the initial occurrence.
2. Should both parameters, namely `start_chr_count` and `occurrence`, be null, the system will yield no output.

USING - `DECODE`

Decode acts as value-by-value substitution. For every value of the field, it checks for a match in a series of if/then tests.

Syntax

```
decode(value, if1, then1, if2, then2, ……. else_)
```

Example

```
SQL> SELECT sal, decode(sal, 500, 'Low', 5000, 'High', 'Medium') FROM emp;

SAL     DECODE
------- -----------
500     Low
2500    Medium
2000    Medium
3500    Medium
3000    Medium
5000    High
4000    Medium
5000    High
```

```
1800    Medium
1200    Medium
2000    Medium
2700    Medium
2200    Medium
3200    Medium

SQL> SELECT decode(1, 1, 3), decode(1, 2, 3, 4, 4, 6) FROM dual;

DECODE(1,1,3) DECODE(1,2,3,4,4,6)
------------- ------------------------
3             6
```

Additional Notes

---

1. In cases where the number of parameters is both odd and non-uniform, the `DECODE` function will yield no result.
2. When the count of parameters is even and varied, the `DECODE` function will display the value associated with the last pair.
3. If all parameters within the `DECODE` function are null, the output will be empty.
4. In scenarios where all parameters are set to zero, the `DECODE` function will display zero.

USING - `GREATEST`

It is used to retrieve the greatest string

Syntax

```
GREATEST(string1, string2, string3, ..., stringN)
```

Example

```
SQL> SELECT GREATEST('a', 'b', 'c'), GREATEST('MEHMET', 'SATRIAN', 'SELAMATH') FROM dual;
```

```
G GREATEST
- --------
c SELAMATH
```

Additional Notes

---

1. In the event that all parameters are null, the function will yield no result.
2. Should any of the parameters be null, the function will produce no output.

USING - `LEAST`

To obtain the least string:

Syntax

```
LEAST(string1, string2, string3, ..., stringN)
```

Example

```
SQL> SELECT LEAST('a', 'b', 'c'), LEAST('MEHMET', 'SATRIAN', 'SELAMATH') FROM dual;
```

```
L LEAST(
- ------
a MEHMET
```

Additional Notes

1. If all parameters are null, the function will not yield any result.
2. If any parameter is null, the function will produce no output.

USING - `COALESCE`

Gives the first non-null string

Syntax

```
coalesce(_strng1, string2, string3 … stringn_)
```

Example

```
SQL> select coalesce('a','b','c'), coalesce(null,'a',null,'b') from dual;
```

```
C C
- -
a a
```

USING - `DATE FUNCTIONS`

Sysdata	Current_data	Current_timestamp	Systimestamp
Localtimestamp	Dbtimezone	Sessiontimezone	To_char
To_date	Add_month	Month_between	Next_day
Last_day	Extract	Greatest	Least
Round	Trunc	New_time	Coalesce

Oracle default date format is DD-MON-YY. We can change the default format to our desired format by using the following command.

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MONTH-YYYY'; -- To set the date format for the session (will expire
when the session is closed)
```

USING - `SYSDATE`

Returns the current date and time.

Syntax

```
SQL> SELECT sysdate FROM dual;
```

Output

```
SYSDATE
-----------
24-DEC-06
```

USING - `CURRENT_DATE`

Returns the current date in the session’s timezone.

Syntax

```
SQL> SELECT current_date FROM dual;
```

```
CURRENT_DATE
------------------
24-DEC-06
```

USING - `CURRENT_TIMESTAMP`

Returns the current timestamp with the active time zone information.

Syntax

```
SQL> SELECT current_timestamp FROM dual;
```

```
CURRENT_TIMESTAMP
---------------------------------------------------
24-DEC-06 03.42.41.383369 AM +05:30
```

USING - `SYSTIMESTAMP`

Returns the system date, including fractional seconds and time zone of the database.

Syntax

```
SQL> SELECT systimestamp FROM dual;
```

```
SYSTIMESTAMP
------------------------------------------------------
24-DEC-06 03.49.31.830099 AM +05:30
```

USING - `LOCALTIMESTAMP`

Returns local timestamp in the active time zone information, with no time zone information shown.

Syntax

```
SQL> SELECT localtimestamp FROM dual;
```

```
LOCALTIMESTAMP
------------------------------------------------------
24-DEC-06 03.44.18.502874 AM
```

USING - `DBTIMEZONE`

Returns the current database time zone in UTC format (Coordinated Universal Time).

Syntax

```
SQL> SELECT dbtimezone FROM dual;
```

```
DBTIMEZONE
---------------
-07:00
```

USING - `SESSIONTIMEZONE`

Returns the value of the current session’s time zone.

Syntax

```
SQL> SELECT sessiontimezone FROM dual;
```

```
SESSIONTIMEZONE
------------------------------------
+05:30
```

Using `TO_CHAR`

Used to extract various date formats.

Syntax

```
to_char(_date_, _format_)
```

DATE FORMATS:

Format	Description
D	No of days in the week
DD	No of days in the month
DDD	No of days in the year
MM	No of the month
MON	Three-letter abbreviation of the month
MONTH	Fully spelled out month
RM	Roman numeral month
DY	Three-letter abbreviated day
DAY	Fully spelled out day
Y	Last one digit of the year
YY	Last two digits of the year
YYY	Last three digits of the year
YYYY	Full four-digit year
SYYYY	Signed year
I	One-digit year from ISO standard
IY	Two-digit year from ISO standard
IYY	Three-digit year from ISO standard
IYYY	Four-digit year from ISO standard
Y, YYY	Year with a comma
YEAR	Fully spelled out year
CC	Century
Q	No of quarters
W	No of weeks in the month
WW	No of weeks in the year
IW	No of weeks in the year from ISO standard
HH	Hours
MI	Minutes
SS	Seconds
FF	Fractional seconds
AM or PM	Displays AM or PM depending upon the time of day
A.M. or P.M.	Displays A.M or P.M depending upon the time of day
AD or BC	Displays AD or BC depending upon the date
A.D or B.C	Displays AD or BC depending upon the date
FM	Prefix to month or day, suppresses padding
TH	Suffix to a number
SP	Suffix to a number to be spelled out
SPTH	Suffix combination of TH and SP to be both spelled out

Format	Description
THSP	Same as SPTH

Examples:

```
-- Example 1
SQL> SELECT TO_CHAR(SYSDATE, 'DD MONTH YYYY HH:MI:SS AM DY') FROM dual;

TO_CHAR(SYSDATE,'DD MONTH YYYYHH:MI'
----------------------------------------------------
24 December 2006 02:03:23 PM Sun

-- Example 2
SQL> SELECT TO_CHAR(SYSDATE, 'DD MONTH YEAR') FROM dual;

TO_CHAR(SYSDATE,'DDMONTHYEAR')
-------------------------------------------------------
24 December Two Thousand Six

-- Example 3
SQL> SELECT TO_CHAR(SYSDATE, 'DD FMMONTH YEAR') FROM dual;

TO_CHAR(SYSDATE,'DD FMMONTH YEAR')
-------------------------------------------------------
24 December Two Thousand Six

-- Example 4
SQL> SELECT TO_CHAR(SYSDATE, 'DDTH DDTH') FROM dual;

TO_CHAR(S
-----------------------------------
24th 24TH

-- Example 5
SQL> SELECT TO_CHAR(SYSDATE, 'DDSPTH DDSPTH') FROM dual;

TO_CHAR(SYSDATE,'DDSPTHDDSPTH'
------------------------------------------
Twenty-Fourth TWENTY-FOURTH

-- Example 6
SQL> SELECT TO_CHAR(SYSDATE, 'DDSP Ddsp DDSP ') FROM dual;

TO_CHAR(SYSDATE,'DDSPDDSPDDSP')
------------------------------------------------
Twenty-Four Twenty-Four TWENTY-FOUR
```

USING - `TO_DATE`

Used to convert the string into a data format.

Syntax

```
to_date(date)
```

Example

```
SQL> SELECT TO_CHAR(TO_DATE('24/DEC/2006','DD/MON/YYYY'), 'DD * MONTH * DAY') FROM dual;
```

```
TO_CHAR(TO_DATE('24/DEC/20'
--------------------------
24 * December * Sunday
```

- If `to_char` isn't used, Oracle will display output in the default date format.

USING - `ADD_MONTHS`

This will add the specified months to the given date.

Syntax

```
add_months(date, no_of_months)
```

Example

```
SQL> SELECT add_months(to_date('11-jan-1990','dd-mon-yyyy'), 5) FROM dual;

ADD_MONTHS
----------------
11-JUN-90

SQL> SELECT add_months(to_date('11-jan-1990','dd-mon-yyyy'), -5) FROM dual;

ADD_MONTH
---------------
11-AUG-89
```

1. If `no_of_months` is zero, it displays the same date.
2. If `no_of_months` is null, it displays nothing.

USING - `MONTHS_BETWEEN`

Provides the difference in months between two dates

Syntax

```
months_between(date1, date2)
```

Example

```
SQL> SELECT months_between(to_date('11-aug-1990', 'dd-mon-yyyy'), to_date('11-jan-1990', 'dd-mon-yyyy')) AS RESULT
FROM dual;

RESULT
------
```

```
7

SQL> SELECT months_between(to_date('11-jan-1990', 'dd-mon-yyyy'), to_date('11-aug-1990', 'dd-mon-yyyy')) AS RESULT
FROM dual;

RESULT
------
-7
```

USING - `NEXT_DAY`

Returns the next day of the given day from the specified date.

Syntax

```
next_day(date, day)
```

Example

```
SQL> SELECT next_day(to_date('24-dec-2006', 'dd-mon-yyyy'), 'sun') FROM dual;

NEXT_DAY(
-------------
31-DEC-06
```

- *If the day parameter is `null`, then it displays nothing.

USING - `LAST_DAY`

This will produce the last day of the given date.

Syntax

```
last_day(date)
```

Example

```
SQL> SELECT last_day(to_date('24-dec-2006', 'dd-mon-yyyy'), 'sun') FROM dual;

LAST_DAY(
-------------
31-DEC-06
```

USING - `EXTRACT`

Used to extract a portion of the date value.

Syntax

```
extract((year | month | day | hour | minute | second), date)
```

Example

```
SQL> SELECT extract(year FROM sysdate) FROM dual;

EXTRACT(YEAR FROM SYSDATE)
--------------------------
2006
```

You can extract only one value at a time.

USING - `GREATEST`

Returns the latest of all the dates.

Syntax

```
greatest(_date1, date2, date3 … daten_)
```

Example

```
SQL > SELECT greatest(
    to_date('11-jan-90', 'dd-mon-yy'),
    to_date('11-mar-90', 'dd-mon-yy'),
    to_date('11-apr-90', 'dd-mon-yy')
) FROM dual;

GREATEST(
-------------
11-APR-90
```

USING - `LEAST`

Returns the earliest among the dates.

Syntax

```
least(_date1, date2, date3 … daten_)
```

Example

```
SQL > SELECT least(
    to_date('11-jan-90', 'dd-mon-yy'),
    to_date('11-mar-90', 'dd-mon-yy'),
    to_date('11-apr-90', 'dd-mon-yy')
) FROM dual;

LEAST(
-------------
11-JAN-90
```

USING - `ROUND`

`ROUND` will round the date to the nearest date that is equal to or greater than the given date.

Syntax

```
round (_date, (_day | month | year_)_)
```

Description

If the second parameter is *`year`*, the `ROUND` function checks the month of the given date in the following ranges:

- `JAN to JUN:` It returns the first day of the current year.
- `JUL to DEC:` It returns the first day of the next year.

If the second parameter is *month*, the `ROUND` function checks the day of the given date in the following ranges:

- `1 to 15:` It returns the first day of the current month.
- `16 to 31:` It returns the first day of the next month.

If the second parameter is *day*, the `ROUND` function checks the weekday of the given date in the following ranges:

- `SUN to WED:` It returns the previous Sunday.
- `THU to SUN:` It returns the next Sunday.

Additional Notes:

1. If the second parameter is null, the function returns nothing.
2. If you do not specify the second parameter, `ROUND` resets the time to the beginning of the current day for user-specified dates.
3. If you do not specify the second parameter, `ROUND` resets the time to the beginning of the next day for `SYSDATE`.

Example

```
SQL> select round(to_date('24-dec-04','dd-mon-yy'),'year'),round(to_date('11-mar- 06','dd-mon-yy'),'year') from
dual;

    ROUND(TO_    ROUND(TO_
    ------------ ---------------
    01-JAN-05     01-JAN-06

SQL> select round(to_date('11-jan-04','dd-mon-yy'),'month'),round(to_date('18- jan-04','dd-mon-yy'),'month') from
dual;

    ROUND(TO_     ROUND(TO_
    ------------- ---------------
    01-JAN-04     01-FEB-04
```

```
SQL> select round(to_date('26-dec-06','dd-mon-yy'),'day'),round(to_date('29-dec-06','dd-mon-yy'),'day') from dual;

   ROUND(TO_ ROUND(TO_

   -------------- --------------

   24-DEC-06 31-DEC-06

SQL> select to_char(round(to_date('24-dec-06','dd-mon-yy')), 'dd mon yyyy hh:mi:ss am') from dual;

   TO_CHAR(ROUND(TO_DATE('

   --------------------------------

   24 dec 2006 12:00:00 am
```

USING - `TRUNC`

It rolls back `date` according to a specified parameter.

Syntax

```
trunc(_date, (_day | month | year_)_)
```

Description

1. If the second parameter were to be *`year`* then it would always return the first day of the current year.
2. If the second parameter were to be *`month`* then it would always return the first day of the current month.
3. If the second parameter were to be *`day`* then it would always return the previous Sunday.
4. If the second parameter were to be `null`, nothing would be returned.
5. If the second parameter were not to be specified then `trunc` would reset the time to the beginning of the current day.

***Note:** It's a lot to remember, but more importantly it's about being careful.*

```
SQL> select trunc(to_date('24-dec-04','dd-mon-yy'),'year'), trunc(to_date('11-mar-06','dd-mon-yy'),'year') from
dual;

    TRUNC(TO_      TRUNC(TO_
    ------------- --------------
    01-JAN-04      01-JAN-06

SQL> select trunc(to_date('11-jan-04','dd-mon-yy'),'month'), trunc(to_date('18-jan- 04','dd-mon-yy'),'month') from
dual;

    TRUNC(TO_      TRUNC(TO_
    ------------- -------------
    01-JAN-04      01-JAN-04

SQL> select trunc(to_date('26-dec-06','dd-mon-yy'),'day'), trunc(to_date('29-dec- 06','dd-mon-yy'),'day') from dual;

     TRUNC(TO_     TRUNC(TO_
    ------------- --------------
     24-DEC-06     24-DEC-06

SQL> select to_char(trunc(to_date('24-dec-06','dd-mon-yy')), 'dd mon yyyy hh:mi:ss am') from dual;

    TO_CHAR(TRUNC(TO_DATE(
    -----------------------
    '24 dec 2006 12:00:00 am'
```

USING - `NEW_TIME`

Returns the desired time zone's date and time.

Syntax

```
new_time(_date, current_timezone, desired_timezone_)
```

USING - `TIME ZONES`

Available time zones are as follows:

Acronyms	Time Zone
AST/ADT	Atlantic Standard/Daylight Time
BST/BDT	Bering Standard/Daylight Time
CST/CDT	Central Standard/Daylight Time
EST/EDT	Eastern Standard/Daylight Time
GMT	Greenwich Mean Time
HST/HDT	Alaska-Hawaii Standard/Daylight Time
MST/MDT	Mountain Standard/Daylight Time
NST	Newfoundland Standard Time
PST/PDT	Pacific Standard/Daylight Time
YST/YDT	Yukon Standard/Daylight Time

Example

```
SQL> select to_char(new_time(sysdate,'gmt','yst'),'dd mon yyyy hh:mi:ss am') from dual;

    TO_CHAR(NEW_TIME(SYSDAT
-----------------------------------

    24 dec 2006 02:51:20 pm

SQL> select to_char(new_time(sysdate,'gmt','est'),'dd mon yyyy hh:mi:ss am') from dual;

    TO_CHAR(NEW_TIME(SYSDAT

    -----------------------

     24 dec 2006 06:51:26 pm
```

USING - `COALESCE`

Returns the first non-null date.

Syntax

```
coalesce(_date1, date2, date3 … daten_)
```

Example

```
SQL> select coalesce('12-jan-90','13-jan-99'), coalesce(null,'12-jan-90','23-mar- 98',null) from dual;

    COALESCE(     COALESCE(
    ------------- ------------

    12-jan-90      12-jan-90
```

USING - `MISCELLANEOUS FUNCTIONS`

Uid	User	Vsize
Dense_rank	Rank	

USING - `UID`

Returns the integer value corresponding to the user currently logged in.

Example

```
SQL> select uid from dual;

 UID
 ----------
 319
```

USING - `USER`

Returns the logged in user's name

Example

```
SQL> select user from dual;

USER
----------------
SELAMATH
```

USING - `VSIZE`

Returns the number of bytes in the expression.

Example

```
SQL> select vsize(123), vsize('computer'), vsize('12-jan-90') from dual;

VSIZE(123)       VSIZE('COMPUTER')     VSIZE('12-JAN-90')
------------- ----------------------- ----------------------
 3                8                      9
```

USING - `RANK`

Returns the non-sequential ranking

Example

```
SQL> select rownum,sal from (select sal from emp order by sal desc);

    ROWNUM        SAL
    ----------    ----------
     1            5000
     2            3000
     3            3000
     4            2975
     5            2850
     6            2450
     7            1600
     8            1500
     9            1300
     10           1250
     11           1250
     12           1100
     13           1000
     14           950
     15           800
```

```
 SQL> select rank(2975) as within group(order by sal desc) from emp;

RANK(2975)WITHINGROUP(ORDERBYSALDESC)

---------------------------------------------------------

 4
```

USING - `DENSE_RANK`

Returns the sequential ranking.

Example

```
SQL> select dense_rank(2975) within group(order by sal desc) from emp;

DENSE_RANK(2975)WITHINGROUP(ORDERBYSALDESC)

-----------------------------------------------------------------

 3
```

USING - `CONVERSION FUNCTIONS`

Bin_to_num	Chartorowid	Rowidtochar	To_number	To_char	To_date

USING - `BIN_TO_NUM`

Converts binary value to its numerical equivalent

Syntax

```
bin_to_num(_binary_bits_)
```

Example

```
SQL> select bin_to_num(1,1,0) from dual;

    BIN_TO_NUM(1,1,0)
    ------------------------
     6
```

***Note:** If you pass in `null` bits then it produces an error.*

USING - `CHARTOROWID`

- Converts a character `string` to act like an internal oracle row identifier or `rowid`.

USING - `ROWIDTOCHAR`

- Converts an internal oracle row identifier or `rowid` to character `string`

USING - `TO_NUMBER`

- Converts a `char` or `varchar` to `number`.

USING - `TO_CHAR`

- Converts a `number` or `date` to character `string`.

USING - `TO_DATE`

- Converts a `number`, `char` or `varchar` to a `date`

USING - `GROUP FUNCTIONS`

Group functions always produce a single output based on it's constraints.

Sum	Avg	Max	Min	Count

USING - `SUM`

Returns the sum of the values in the specified column.

Syntax

```
sum (_column_)
```

Example

```
SQL> select sum(sal) from emp;

 SUM(SAL)

 ----------

 38600
```

USING - `AVG`

Returns the average of the values in the specified column

Syntax

```
avg(_column_)
```

Example

```
SQL> select avg(sal) as 'AVERAGE_SALARY' from emp;

 AVERAGE_SALARY
 ---------------
 2757.14286
```

USING - `MAX`

Returns the maximum of the values in the specified column

Syntax

```
max(_column_)
```

Example

```
SQL> select max(sal) from emp;

 MAX(SAL)
----------
 5000
```

USING - `MIN`

Returns the minimum of the values in the specified column.

Syntax

```
min (_column_)
```

Example

```
SQL> select min(sal) from emp;

 MIN(SAL)

 ----------

 500
```

USING - `COUNT`

Returns the count of the values of the specified column

Syntax

```
count(_column_)
```

Example

```
SQL> select count(sal),count(*) from emp;

    COUNT(SAL)     COUNT(*)

    -------------- ------------

     14             14
```

End