FUNCTIONS

FUNCTIONS

Oracle has many built-in functions. They are categorized into:

- Numeric/Number Functions
- String Functions
- Null Functions
- Date Functions
- Conversion Functions
- Aggregate Functions

SAMPLE RELATION: Temperature_Range

CITY	COUNTRY	HIGH	LOW	RANGE
Athens	Greece	31.7	5.0	
Bangkok	Thailand	35.0	20.6	14.4
Beijing	China	30.1	-8.3	
Berlin	Germany	23.9	-3.9	
Belgrade	Serbia	26.7	-1.7	28.3

NUMBER FUNCTIONS

NUMBER FUNCTIONS

These are functions that accept numeric input and return numeric values.

- **ABS**
- **SIGN**
- **CEIL**
- **FLOOR**
- **GREATEST**
- **LEAST**
- **POWER**
- **SQRT**
- **MOD**
- **ROUND**
- **TRUNC**
- **MEDIAN**
- **ROWNUM**
- **EXP**

NUMBER FUNCTIONS

ABS

Returns the absolute value of a number.

```
SQL> select ABS(-23.36) from DUAL;

ABS(-23.36)
-----------
      23.36

SQL> select ABS(23.45) from DUAL;

ABS(23.45)
----------
     23.45
```

Display the Absolute value of low temperature of all cities.

```
SQL> select ABS(LOW) from Temperature_Range;

  ABS(LOW)
----------
        50
      20.6
       8.3
       3.9
       1.7
```

NUMBER FUNCTIONS

SIGN

Returns a value indicating the sign of a number.

```
SQL> select SIGN(56) from DUAL;
  SIGN(56)
----------
         1
SQL> select SIGN(-56) from DUAL;
 SIGN(-56)
----------
        -1
```

Display the sign of low temperature of all cities.

```
SQL> select SIGN(LOW) from Temperature_Range;
 SIGN(LOW)
----------
         1
         1
        -1
        -1
        -1
```

NUMBER FUNCTIONS

CEIL

Returns the smallest integer value that is greater than or equal to a number.

```
SQL> select CEIL(56.23) from DUAL;

CEIL(56.23)
-----------
         57

SQL> select CEIL(-56.23) from DUAL;

CEIL(-56.23)
------------
         -56
```

Display the ceil value of high temperature of all cities.

```
SQL> select CEIL(HIGH) from Temperature_Range;

CEIL(HIGH)
----------
        32
        35
        31
        24
        27
```

NUMBER FUNCTIONS

FLOOR

Returns the largest integer value that is equal to or less than a number.

```
SQL> select FLOOR(56.23) from DUAL;

FLOOR(56.23)
------------
          56

SQL> select FLOOR(-56.23) from DUAL;

FLOOR(-56.23)
-------------
          -57
```

Display the floor value of high temperature of all cities.

```
SQL> select FLOOR(HIGH) from Temperature_Range;

FLOOR(HIGH)
-----------
         31
         35
         30
         23
         26
```

NUMBER FUNCTIONS

GREATEST

Returns the greatest value in a list of expressions.

```
SQL> select GREATEST('a','F','z',1) from dual;

G
-
z

SQL> select GREATEST('a','F','z','G') from dual;

G
-
z

SQL> select GREATEST('apple','applis','applas') from dual;

GREATE
------
applis
```

NUMBER FUNCTIONS

GREATEST

```
SQL> select Greatest(City,Country) from Temperature_Range;

GREATEST(CIT
------------
Greece
Thailand
China
Germany
serbia

SQL> select Greatest(High,Low) from Temperature_Range;

GREATEST(HIGH,LOW)
------------------
                50
                35
              30.1
              23.9
              26.7
```

NUMBER FUNCTIONS

LEAST

Returns the smallest value in a list of expressions.

```
SQL> select LEAST('a','F','z',1) from dual;

L
-
1

SQL> select LEAST('a','F','z','G') from dual;

L
-
F

SQL> select LEAST('apple','applis','applas') from dual;

LEAST(
------
applas
```

NUMBER FUNCTIONS

LEAST

```
SQL> select Least(City,Country) from Temperature_Range;

LEAST(CITY,C
------------
Athens
Bangkok
Beijing
Berlin
Belgrade

SQL> select Least(High,Low) from Temperature_Range;

LEAST(HIGH,LOW)
---------------
           31.7
           20.6
           -8.3
           -3.9
           -1.7
```

NUMBER FUNCTIONS

POWER

Returns m raised to the nth power.

```
SQL> select POWER(5,2) from DUAL;

POWER(5,2)
----------
        25

SQL> select POWER(-5,2) from DUAL;

POWER(-5,2)
-----------
         25

SQL> select POWER(5.3,2) from DUAL;

POWER(5.3,2)
------------
       28.09

SQL> select POWER(RANGE,3) from Temperature_Range;

POWER(RANGE,3)
--------------

      2985.984

     23639.903
```

NUMBER FUNCTIONS

SQRT

Returns the square root of a number.

```
SQL> select SQRT(100) from DUAL;

 SQRT(100)
----------
        10

SQL> select SQRT(-100) from DUAL;
select SQRT(-100) from DUAL
           *
ERROR at line 1:
ORA-01428: argument '-100' is out of range


SQL> select SQRT(37) from DUAL;

  SQRT(37)
----------
6.08276253

SQL> select SQRT(HIGH) from Temperature_Range;

SQRT(HIGH)
----------
 5.6302753
5.91607978
5.48634669
4.88876262
5.16720427
```

The argument passed inside a SQRT function should be a positive number.

NUMBER FUNCTIONS

MOD

Returns the remainder of n divided by m.

```
SQL> select MOD(25,3) from DUAL;

 MOD(25,3)
----------
         1

SQL> select MOD(26,3) from DUAL;

 MOD(26,3)
----------
         2

SQL> select MOD(LOW,3) from Temperature_Range;

MOD(LOW,3)
----------
         2
       2.6
      -2.3
        -.9
      -1.7
```

NUMBER FUNCTIONS

ROUND

Returns a number rounded to a certain number of decimal places.

```
SQL> select ROUND(123.56) from DUAL;

ROUND(123.56)
-------------
          124

SQL> select ROUND(123.21) from DUAL;

ROUND(123.21)
-------------
          123

SQL> select ROUND(123.01) from DUAL;

ROUND(123.01)
-------------
          123
```

NUMBER FUNCTIONS

ROUND

```
SQL> select ROUND(236.156,1) from DUAL;

ROUND(236.156,1)
----------------
           236.2

SQL> select ROUND(236.156,2) from DUAL;

ROUND(236.156,2)
----------------
          236.16

SQL> select ROUND(236.156,-1) from DUAL;

ROUND(236.156,-1)
-----------------
              240

SQL> select ROUND(LOW,1) from temperature_range;

ROUND(LOW,1)
------------
          50
        20.6
        -8.3
        -3.9
        -1.7
```

NUMBER FUNCTIONS

TRUNC

Returns a number truncated to a certain number of decimal places.

```
SQL> select TRUNC(236.156,1) from DUAL;

TRUNC(236.156,1)
----------------
           236.1

SQL> select TRUNC(236.156,2) from DUAL;

TRUNC(236.156,2)
----------------
          236.15

SQL> select TRUNC(236.156,-1) from DUAL;

TRUNC(236.156,-1)
-----------------
              230

SQL> select TRUNC(LOW,1) from temperature_range;

TRUNC(LOW,1)
------------
          50
        20.6
        -8.3
        -3.9
        -1.7
```

NUMBER FUNCTIONS

DIFFERENCE BETWEEN TRUNC AND DELETE

- TRUNC and ROUND function looks similar but not exactly.
- ROUND function used to **round the number to the nearest** while TRUNC used to **truncate/delete the number from some position**. Some cases both returns same result.

NUMBER FUNCTIONS

MEDIAN

Returns the median value of an expression.

Syntax:

MEDIAN(Column_Name)

```
SQL> select MEDIAN(5,4,6) from DUAL;
select MEDIAN(5,4,6) from DUAL
       *
ERROR at line 1:
ORA-00909: invalid number of arguments


SQL> select MEDIAN(HIGH) from Temperature_Range;

MEDIAN(HIGH)
------------
        30.1
```

NUMBER FUNCTIONS

ROWNUM

Returns a number that represents the order that a row is.

```
SQL> select ROWNUM, Temperature_Range.* from Temperature_Range;

    ROWNUM CITY           COUNTRY          HIGH         LOW       RANGE
---------- ------------ ---------- ---------- ---------- ----------
         1 Athens       Greece             31.7         50
         2 Bangkok      Thailand             35       20.6       14.4
         3 Beijing      China              30.1       -8.3
         4 Berlin       Germany            23.9       -3.9
         5 Belgrade     serbia             26.7       -1.7       28.7

SQL> select ROWNUM, Temperature_Range.* from Temperature_Range where Range IS NULL;

    ROWNUM CITY           COUNTRY          HIGH         LOW       RANGE
---------- ------------ ---------- ---------- ---------- ----------
         1 Athens       Greece             31.7         50
         2 Beijing      China              30.1       -8.3
         3 Berlin       Germany            23.9       -3.9

SQL> select ROWNUM, Temperature_Range.* from Temperature_Range where High > 30;

    ROWNUM CITY           COUNTRY          HIGH         LOW       RANGE
---------- ------------ ---------- ---------- ---------- ----------
         1 Athens       Greece             31.7         50
         2 Bangkok      Thailand             35       20.6       14.4
         3 Beijing      China              30.1       -8.3
```

There are no parameters or arguments for the ROWNUM function. The ROWNUM function is sometimes referred to as a **pseudo column** in Oracle.

NUMBER FUNCTIONS

EXP

Returns e raised to the power of number.

```
SQL> select EXP(5.23) from DUAL;

 EXP(5.23)
----------
186.792804

SQL> select EXP(15.23) from DUAL;

EXP(15.23)
----------
 4114385.3

SQL> select EXP(LOW) from Temperature_Range;

  EXP(LOW)
----------
5.1847E+21
 884028624
.000248517
.020241911
.182683524
```

STRING FUNCTIONS

STRING FUNCTIONS

Character or string functions are used to manipulate text strings.

They accept strings or characters as input and can return both character and number values as output.

- **ASCII**
- **CHR**
- **LENGTH**
- **LOWER**
- **UPPER**
- **INITCAP**
- **LPAD**
- **RPAD**
- **TRANSLATE**
- **REPLACE**
- **TRIM**
- **LTRIM**
- **RTRIM**
- **SUBSTR**
- **INSTR**
- **REVERSE**
- **CONCAT**

STRING FUNCTIONS

ASCII

Returns the NUMBER code that represents the specified character.

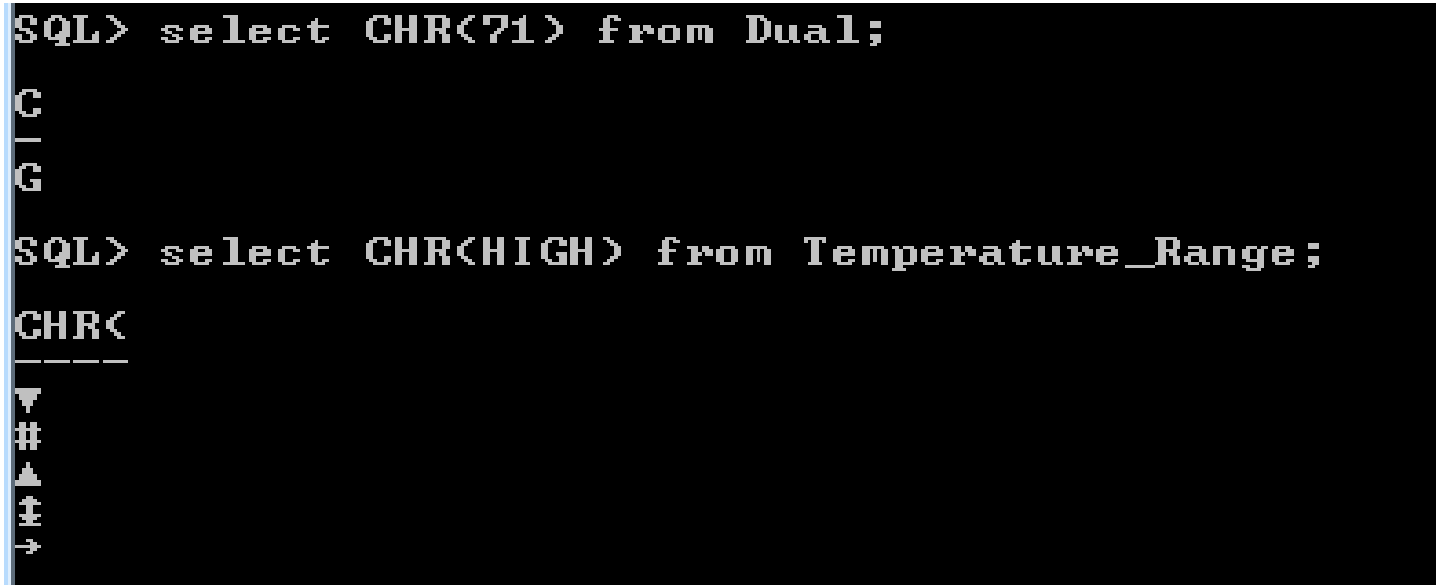
```
SQL> select ASCII('G') from Dual;

ASCII('G')
----------
        71

SQL> select ASCII(City) from Temperature_Range;

ASCII(CITY)
-----------
         65
         66
         66
         66
         66
```

Here in second example the ASCII function returned the value for the first character of the City name and ignored rest of the characters.

STRING FUNCTIONS

CHR

CHR is the opposite of the ASCII function. It returns the character based on the NUMBER code.



STRING FUNCTIONS

LENGTH

Returns the length of the specified string.

```
SQL> select LENGTH(null) from Dual;
LENGTH(NULL)
------------

SQL> select LENGTH(' ') from Dual;
LENGTH('')
----------
         1
SQL> select LENGTH('DBMS') from Dual;
LENGTH('DBMS')
--------------
             4
SQL> select LENGTH(city) from Temperature_Range;
LENGTH(CITY)
------------
           6
           7
           7
           6
           8
```

If the **input string is NULL**, then the LENGTH function will **return NULL**.

STRING FUNCTIONS

LOWER

Converts all letters in the specified string to lowercase.

```
SQL> select LOWER('DBMS') from DUAL;

LOWE
----
dbms

SQL> select LOWER('DBMS 123') from DUAL;

LOWER('D
--------
dbms 123

SQL> select LOWER(Country) from Temperature_Range;

LOWER(COUN
----------
greece
thailand
china
germany
serbia
```

If there are characters in the string that are not letters, they are unaffected by this function.

STRING FUNCTIONS

UPPER

Converts all letters in the specified string to uppercase.

```
SQL> select UPPER('dbms') from DUAL;

UPPE
----
DBMS

SQL> select UPPER('dbms 123') from DUAL;

UPPER('D
--------
DBMS 123

SQL> select UPPER('dBMs 123') from DUAL;

UPPER('D
--------
DBMS 123

SQL> select UPPER(Country) from Temperature_Range;

UPPER(COUN
----------
GREECE
THAILAND
CHINA
GERMANY
SERBIA
```

If there are characters in the string that are not letters, they are unaffected by this function.

STRING FUNCTIONS

INITCAP

Sets the first character in each word to uppercase and the rest to lowercase.

```
SQL> select INITCAP('database management systems') from DUAL;

INITCAP('DATABASEMANAGEMENT
---------------------------
Database Management Systems

SQL> select INITCAP(Country) from Temperature_Range;

INITCAP(CO
----------
Greece
Thailand
China
Germany
Serbia
```

STRING FUNCTIONS

LPAD

Pads the left-side of a string with a specific set of characters (when input string is not null).

Syntax:

LPAD(input string, padding_length, [pad_string])

Parameters or Arguments:

- padded_length - The number of characters to return. If the padding_length is smaller than the original string, the LPAD function will truncate the string to the size of padding_length.
- pad_string - Optional. This is the string that will be padded to the left-hand side of input string. If this parameter is omitted, the LPAD function will pad spaces to the left-side of input string.

STRING FUNCTIONS

LPAD

```
SQL> select LPAD('DATABASE',12) from dual;

LPAD('DATABA
------------
    DATABASE

SQL> select LPAD('DATABASE',12,'*') from dual;

LPAD('DATABA
------------
****DATABASE

SQL> select LPAD('DATABASE',1) from dual;

L
-
D

SQL> select LPAD(COUNTRY,15,'*') from Temperature_Range;

LPAD(COUNTRY,15,'*')
--------------------------------------------------
**********Greece
*********Thailand
************China
*********Germany
**********serbia
```

STRING FUNCTIONS

RPAD

Pads the right-side of a string with a specific set of characters (when input string is not null).

Syntax:

RPAD(input string, padding_length, [pad_string])

Parameters or Arguments:

- padded_length - The number of characters to return. If the padding_length is smaller than the original string, the RPAD function will truncate the string to the size of padding_length.
- pad_string - Optional. This is the string that will be padded to the right-hand side of input string. If this parameter is omitted, the RPAD function will pad spaces to the right-side of input string.

STRING FUNCTIONS

RPAD

```
SQL> select RPAD('DATABASE',12) from dual;

RPAD('DATABA
------------
DATABASE

SQL> select RPAD('DATABASE',12,'*') from dual;

RPAD('DATABA
------------
DATABASE****

SQL> select RPAD('DATABASE',1) from dual;

R
-
D

SQL> select RPAD(COUNTRY,15,'*') from Temperature_Range;

RPAD(COUNTRY,15,'*')
--------------------------------------------------------------------------------
Greece**********
Thailand********
China***********
Germany*********
serbia**********
```

STRING FUNCTIONS

TRANSLATE

Replaces a sequence of characters in a string with another set of characters. However, it replaces a single character at a time.

For example, it will replace the 1st character in the string_to_replace with the 1st character in the replacement_string. Then it will replace the 2nd character in the string_to_replace with the 2nd character in the replacement_string, and so on.

Syntax:

TRANSLATE(Input_String, string_to_replace, replacement_string)

Parameters or Arguments:

string_to_replace - The string that will be searched for in string1.

replacement_string - All characters in the string_to_replace will be replaced with the corresponding character in the replacement_string.

STRING FUNCTIONS

TRANSLATE

```
SQL> select TRANSLATE('CSE A,B DBMS','ABC','DEF') from DUAL;

TRANSLATE('C
------------
FSE D,E DEMS

SQL> select TRANSLATE(COUNTRY,'er','12') from Temperature_Range;

TRANSLATE(COUNTRY,'ER','12')
------------------------------------
G211c1
Thailand
China
G12many
s12bia

SQL> select TRANSLATE('CSE A,B DBMS','ABC') from DUAL;
select TRANSLATE('CSE A,B DBMS','ABC') from DUAL
       *
ERROR at line 1:
ORA-00909: invalid number of arguments
```

If the replacement_string parameter is omitted, it throws error.

STRING FUNCTIONS

REPLACE

Replaces a sequence of characters in a string with another set of characters.

Syntax:

REPLACE(string1, string_to_replace [,replacement_string])

Parameters or Arguments:

string_to_replace - The string that will be searched for in string1.

replacement_string - Optional. All occurrences of string_to_replace will be replaced with replacement_string in string1.

STRING FUNCTIONS

REPLACE

```
SQL> select REPLACE('CSE A,B DBMS','CSE','EIE') from DUAL;

REPLACE('CSE
------------
EIE A,B DBMS

SQL> select REPLACE(COUNTRY,'er','12') from Temperature_Range;

REPLACE(COUNTRY,'ER'
--------------------
Greece
Thailand
China
G12many
s12bia

SQL> select REPLACE(COUNTRY,'er') from Temperature_Range;

REPLACE(CO
----------
Greece
Thailand
China
Gmany
sbia
```

If the replacement_string parameter is omitted, the REPLACE function simply removes all occurrences of string_to_replace and returns the resulting string.

STRING FUNCTIONS

DIFFERENCE BETWEEN TRANSLATE AND REPLACE

TRANSLATE	REPLACE
Translate function searches for a character and it replaces in occurrence of the character.	Replace function searches for a string and replaces with the given string.
Returns null if no match found	Returns string if no match found
Replaces character one-to-one basis	Replaces entire string at a time

STRING FUNCTIONS

TRIM

Removes all specified characters either from the beginning or the end of a string.

Syntax:

TRIM(LEADING|TRAILING | BOTH trim_character FROM Input_String**)**

Example:

```
SQL> select TRIM('  DATABASE  ') from Dual;

TRIM('DA
--------
DATABASE

SQL> select TRIM(' ' FROM '  DATABASE  ') from Dual;

TRIM(''F
--------
DATABASE
```

STRING FUNCTIONS

TRIM

```
SQL> select TRIM(LEADING '0' FROM '00DATA0BASE00') from Dual;

TRIM(LEADIN
-----------
DATA0BASE00

SQL> select TRIM(TRAILING '0' FROM '00DATA0BASE00') from Dual;

TRIM(TRAILI
-----------
00DATA0BASE

SQL> select TRIM(BOTH '0' FROM '00DATA0BASE00') from Dual;

TRIM(BOTH
---------
DATA0BASE

SQL> select TRIM('0' FROM '00DATA0BASE00') from Dual;

TRIM('0'F
---------
DATA0BASE
```

STRING FUNCTIONS

TRIM

```
SQL> select TRIM(LEADING '0' FROM '00DATA0BASE00') from Dual;

TRIM(LEADIN
-----------
DATA0BASE00

SQL> select TRIM(TRAILING '0' FROM '00DATA0BASE00') from Dual;

TRIM(TRAILI
-----------
00DATA0BASE

SQL> select TRIM(BOTH '0' FROM '00DATA0BASE00') from Dual;

TRIM(BOTH
---------
DATA0BASE

SQL> select TRIM('0' FROM '00DATA0BASE00') from Dual;

TRIM('0'F
---------
DATA0BASE
```

STRING FUNCTIONS

TRIM

```
SQL> select TRIM(LEADING 'B' FROM City) from Temperature_Range;

TRIM(LEADING
------------
Athens
angkok
eijing
erlin
elgrade

SQL> select TRIM(TRAILING 'a' FROM Country) from Temperature_Range;

TRIM(TRAIL
----------
Greece
Thailand
Chin
Germany
serbi

SQL> select TRIM('G' FROM Country) from Temperature_Range;

TRIM('G'FR
----------
reece
Thailand
China
ermany
serbia
```

If you do not choose a value for the first parameter (LEADING, TRAILING, BOTH), the TRIM function will remove trim_character from both the front and end of input string.

STRING FUNCTIONS

LTRIM

Removes all specified characters from the left-hand side of a string.

Syntax:

LTRIM(Input_string [, trim_string])

```
SQL> select LTRIM('123DATABASE123','12') from Dual;

LTRIM('123DA
------------
3DATABASE123

SQL> select LTRIM('  123DATABASE123') from Dual;

LTRIM('123DATA
--------------
123DATABASE123
```

STRING FUNCTIONS

LTRIM

```
SQL> select LTRIM('12312DATABASE123','12') from Dual;

LTRIM('12312DA
--------------
312DATABASE123

SQL> select LTRIM('12312DATABASE123','124') from Dual;

LTRIM('12312DA
--------------
312DATABASE123
```

```
SQL> select LTRIM(City,'A') from Temperature_Range;

LTRIM(CITY,'
------------
thens
Bangkok
Beijing
Berlin
Belgrade

SQL> select LTRIM(City,'Be') from Temperature_Range;

LTRIM(CITY,'
------------
Athens
angkok
ijing
rlin
lgrade
```

STRING FUNCTIONS

RTRIM

Removes all specified characters from the right-hand side of a string.

Syntax:

RTRIM(Input_String [, trim_string])

```
SQL> select RTRIM('  123DATABASE123  ') from Dual;

RTRIM('123DATABA
----------------
  123DATABASE123

SQL> select RTRIM('12312DATABASE123','12') from Dual;

RTRIM('12312DATA
----------------
12312DATABASE123

SQL> select RTRIM('12312DATABASE12','12') from Dual;

RTRIM('12312D
-------------
12312DATABASE
```

STRING FUNCTIONS

RTRIM

```
SQL> select RTRIM('12312DATABASE12','12') from Dual;

RTRIM('12312D
-------------
12312DATABASE

SQL> select RTRIM('12312DATABASE12','124') from Dual;

RTRIM('12312D
-------------
12312DATABASE
```

```
SQL> select RTRIM(City,'g') from Temperature_Range;

RTRIM(CITY,'
------------
Athens
Bangkok
Beijin
Berlin
Belgrade

SQL> select RTRIM(City,'gn') from Temperature_Range;

RTRIM(CITY,'
------------
Athens
Bangkok
Beiji
Berli
Belgrade
```

STRING FUNCTIONS

SUBSTR

Allows you to extract a substring from a string.

Syntax:

SUBSTR(Input_string, start_position [, length])

```
SQL> select SUBSTR('Database Management Systems',5) from DUAL;

SUBSTR('DATABASEMANAGEM
-----------------------
base Management Systems

SQL> select SUBSTR('Database Management Systems',3,4) from DUAL;

SUBS
----
taba

SQL> select SUBSTR('Database Management Systems',-6,4) from DUAL;

SUBS
----
yste
```

STRING FUNCTIONS

SUBSTR

```
SQL> select SUBSTR('Database Management Systems',-3,4) from DUAL;

SUB
---
ems

SQL> select SUBSTR(Country,2) from Temperature_Range;

SUBSTR(COUNTRY,2)
------------------------------------
reece
hailand
hina
ermany
erbia

SQL> select SUBSTR(Country,3,2) from Temperature_Range;

SUBSTR(C
--------
ee
ai
in
rm
rb

SQL> select SUBSTR(Country,-3,2) from Temperature_Range;

SUBSTR(C
--------
ec
an
in
an
bi
```

STRING FUNCTIONS

INSTR

Returns the location of a substring in a string.

Syntax:

INSTR(Input_string, [, start_position [,th_appearance]])

```
SQL> select INSTR('Database Management Systems','e') from DUAL;

INSTR('DATABASEMANAGEMENTSYSTEMS','E')
--------------------------------------
                                     8

SQL> select INSTR('Database Management Systems','e',1) from DUAL;

INSTR('DATABASEMANAGEMENTSYSTEMS','E',1)
----------------------------------------
                                       8

SQL> select INSTR('Database Management Systems','e',2) from DUAL;

INSTR('DATABASEMANAGEMENTSYSTEMS','E',2)
----------------------------------------
                                       8
```

STRING FUNCTIONS

INSTR

```
SQL> select INSTR('Database Management Systems','e',2,1) from DUAL;

INSTR('DATABASEMANAGEMENTSYSTEMS','E',2,1)
------------------------------------------
                                         8

SQL> select INSTR('Database Management Systems','e',5,2) from DUAL;

INSTR('DATABASEMANAGEMENTSYSTEMS','E',5,2)
------------------------------------------
                                        15
```

```
SQL> select INSTR('Database Management Systems','e',-3,1) from DUAL;

INSTR('DATABASEMANAGEMENTSYSTEMS','E',-3,1)
-------------------------------------------
                                         25

SQL> select INSTR('Database Management Systems','e',-4,2) from DUAL;

INSTR('DATABASEMANAGEMENTSYSTEMS','E',-4,2)
-------------------------------------------
                                         15
```

If the substring is not found in the input string, INSTR will return 0.

STRING FUNCTIONS

INSTR

```
SQL> select INSTR(Country,'e') from Temperature_Range;
INSTR(COUNTRY,'E')
------------------
                 3
                 0
                 0
                 2
                 2

SQL> select INSTR(Country,'e',2) from Temperature_Range;
INSTR(COUNTRY,'E',2)
--------------------
                   3
                   0
                   0
                   2
                   2

SQL> select INSTR(Country,'e',2,2) from Temperature_Range;
INSTR(COUNTRY,'E',2,2)
----------------------
                     4
                     0
                     0
                     0
                     0

SQL> select INSTR(Country,'e',-2,2) from Temperature_Range;
INSTR(COUNTRY,'E',-2,2)
-----------------------
                      3
                      0
                      0
                      0
                      0
```

STRING FUNCTIONS

REVERSE

It is used to reverse the given input string.

```
SQL> select REVERSE('Database Management Systems') from DUAL;

REVERSE('DATABASEMANAGEMENT
---------------------------
smetsyS tnemeganaM esabataD
```

```
SQL> select REVERSE(Country) from Temperature_Range;

REVERSE(CO
----------
eceerG
dnaliahI
anihC
ynamreG
aibres
```

STRING FUNCTIONS

CONCAT

It is used to merge or join two strings.

```
SQL> select CONCAT('Database','Management') from DUAL;

CONCAT('DATABASE',
------------------
DatabaseManagement
```

```
SQL> select CONCAT('Database','Management','Systems') from DUAL;
select CONCAT('Database','Management','Systems') from DUAL
       *
ERROR at line 1:
ORA-00909: invalid number of arguments
```

STRING FUNCTIONS

CONCAT

```
SQL> select CONCAT(CONCAT('Database','Management'),'Systems') from DUAL;

CONCAT(CONCAT('DATABASE',
------------------------
DatabaseManagementSystems
```

```
SQL> select CONCAT(Country,City) from Temperature_Range;

CONCAT(COUNTRY,CITY)
--------------------
GreeceAthens
ThailandBangkok
ChinaBeijing
GermanyBerlin
serbiaBelgrade
```

NULL FUNCTIONS

NULL FUNCTIONS

Null functions can translate NULL into a value and inversely. Any data type can contain NULL: number, date and varchar.

- **NVL**
- **NVL2**
- **NULLIF**
- **DECODE**
- **COALESCE**
- **CASE**

NULL FUNCTIONS

NVL

Substitutes a value when a null value is encountered.

Syntax:

NVL(Column_Name, Value_to_replace)

Example:

select NVL(Range, 0) from Temperature_Range;

This SQL statement would return 0 if the Range field contained a null value. Otherwise, it would return the value in Range field.

```
SQL> select NVL(Range,0) from Temperature_Range;

NVL(RANGE,0)
------------
           0
        14.4
           0
           0
        28.7
```

NULL FUNCTIONS

NVL

select City, NVL(Range, High) from Temperature_Range;

The statement would return the High field if the Range contained a null value. Otherwise, it would return the value in Range field.

```
SQL> select City,NVL(Range,High) from Temperature_Range;

CITY          NVL(RANGE,HIGH)
------------- ---------------
Athens                   31.7
Bangkok                  14.4
Beijing                  30.1
Berlin                   23.9
Belgrade                 28.7
```

NULL FUNCTIONS

NVL2

- Extends the functionality found in the NVL function.
- Substitutes a value when a null value is encountered as well as when a non-null value is encountered.

Syntax:

NVL2(Coulmn_Name, value_if_not_null, value_if_null)

Example:

select NVL2(Range, 'NOT NULL', ‘NULL') from Temperature_Range;

This statement would return ‘NULL' if the Range field contained a null value. Otherwise, it would return 'NOT NULL'.

NULL FUNCTIONS

NVL2

```
SQL> select NVL2(Range,'NOT NULL','NULL') from Temperature_Range;

NVL2(RAN
--------
NULL
NOT NULL
NULL
NULL
NOT NULL
```

select City, NVL2(Range, City, 0) from Temperature_Range;

The statement would return the City field if the Range contained a null value. Otherwise, it would return 0.

```
SQL> select City,NVL2(Range,City,0) from Temperature_Range;

CITY          NVL2(RANGE,C
------------- -------------
Athens        0
Bangkok       Bangkok
Beijing       0
Berlin        0
Belgrade      Belgrade
```

NULL FUNCTIONS

NULLIF

Compares expr1 and expr2. If expr1 and expr2 are equal, the NULLIF function returns NULL. Otherwise, it returns expr1.

Syntax:

NULLIF(expr1, expr2)

Note:

Datatype of expr1 and expr2 should be same.

```
SQL> select NULLIF(12,12) from dual;

NULLIF(12,12)
-------------


SQL> select NULLIF(12,14) from dual;

NULLIF(12,14)
-------------
           12
```

NULL FUNCTIONS

NULLIF

```
SQL> select NULLIF('a','b') from dual;

N
-
a

SQL> select NULLIF('Apples','apples') from dual;

NULLIF
------
Apples
```

```
SQL> select NULLIF(High,Low) from Temperature_Range;

NULLIF(HIGH,LOW)
----------------
            31.7
              35
            30.1
            23.9
            26.7
```

NULL FUNCTIONS

COALESCE

- Returns the first non-null expression in the list.
- If all expressions evaluate to null, then the COALESCE function will return null.

Syntax:

COALESCE(expr1, expr2, ... expr_n)

Note:

Datatype of all expressions should be same.

```
SQL> select COALESCE('Apples','apples',NULL) from dual;

COALES
------
Apples

SQL> select COALESCE(23,58,94,NULL,12) from dual;

COALESCE(23,58,94,NULL,12)
--------------------------
                        23
```

NULL FUNCTIONS

COALESCE

```
SQL> select COALESCE(23+Null,NULL,12) from dual;

COALESCE(23+NULL,NULL,12)
-------------------------
                       12

SQL> select COALESCE(Null,NULL) from dual;

C
-
```

```
SQL> select COALESCE(High,Low,Range) from Temperature_Range;

COALESCE(HIGH,LOW,RANGE)
------------------------
                    31.7
                      35
                    30.1
                    23.9
                    26.7
```

NULL FUNCTIONS

DECODE

Has the functionality of an IF-THEN-ELSE statement.

Syntax:

DECODE(expression , search , result [, search , result]... [, default])

Parameters or Arguments:

Expression ➔ The value to compare.

Search ➔ The value that is compared against expression.

Result ➔ The value returned, if expression is equal to search.

Default ➔ Optional. If no matches are found, the DECODE function will return default. If default is omitted, then the DECODE function will return null (if no matches are found).

NULL FUNCTIONS

DECODE

```
SQL> select DECODE(High,31.7,'Very High',26.7,'Medium',23,'Low','No Prediction')
 from Temperature_Range;

DECODE(HIGH,3
-------------
Very High
No Prediction
No Prediction
No Prediction
Medium
```

The above query is equivalent to:

```
IF High = 31.7 THEN
        result := 'Very High';
ELSIF High = 26.7 THEN
        result := 'Medium';
ELSE
        result := 'No Prediction';
END IF;
```

NULL FUNCTIONS

```
SQL> select Temperature_Range.*,DECODE(Range,14.4,'Mentioned')AS RESULT from Tem
perature_Range;

CITY         COUNTRY          HIGH        LOW      RANGE RESULT
------------ ---------- ---------- ---------- ---------- ---------
Athens       Greece           31.7         50
Bangkok      Thailand           35       20.6       14.4 Mentioned
Beijing      China            30.1       -8.3
Berlin       Germany          23.9       -3.9
Belgrade     serbia           26.7       -1.7       28.7
```

The above query is equivalent to:

IF Range = 14.4 THEN

result := ‘Mentioned';

END IF;

[Here the default value is not mentioned in the query, hence the **DECODE** function returned **NULL**]

NULL FUNCTIONS

CASE

Has the functionality of an IF-THEN-ELSE statement.

Syntax:

```
CASE [ expression ]
        WHEN condition_1 THEN result_1
        WHEN condition_2 THEN result_2
        ...
        WHEN condition_n THEN result_n
        ELSE result
END
```

Parameters or Arguments:

Expression ➔ Optional. It is the value that you are comparing to the list of conditions. condition_1, **condition_2, ... condition_n ➔** The conditions are evaluated in the order listed. Once a condition is found to be true, the CASE statement will return the result and not evaluate the conditions any further.

result_1, result_2, ... result_n ➔ This is the value returned once a condition is found to be true.

NULL FUNCTIONS

CASE

- **Value Match (Simple) CASE Expression**

```
SELECT ename, empno, deptno,
(CASE deptno WHEN 10 THEN 'Accounting'
WHEN 20 THEN 'Research'
WHEN 30 THEN 'Sales'
WHEN 40 THEN 'Operations'
ELSE 'Unknown' END) department FROM emp;
```

- **Searched CASE Expression**

```
SELECT ename, empno, sal,
(CASE
WHEN sal < 1000 THEN 'Low'
WHEN sal BETWEEN 1000 AND 3000 THEN 'Medium'
WHEN sal > 3000 THEN 'High' ELSE 'N/A' END) salary FROM emp;
```

NULL FUNCTIONS

CASE

```
SQL> select
  2     CASE
  3         WHEN 'a'<'b' THEN 'Hello'
  4         WHEN 'd'>'e' THEN 'HI'
  5     END
  6   from DUAL;

CASEW
------
Hello
```

```
SQL> select
  2     CASE
  3         WHEN High>30 THEN 'EXTREME'
  4         WHEN High<30 THEN 'MODERATE'
  5         ELSE 'NOT MENTIONED'
  6     END
  7  from Temperature_Range;

CASEWHENHIGH>
--------------
EXTREME
EXTREME
EXTREME
MODERATE
MODERATE
```

DATE FUNCTIONS

DATE FUNCTIONS

These are functions that take values that are of DATE datatype as input and return values of DATE datatype, except for the MONTHS_BETWEEN function, which returns a number.

- **SYSDATE**
- **CURRENT_DATE**
- **SYSTIMESTAMP**
- **CURRENT_TIMESTAMP**
- **LOCALTIMESTAMP**
- **ADD_MONTHS**
- **MONTHS_BETWEEN**
- **NEXT_DAY**
- **LAST_DAY**
- **EXTRACT**
- **ROUND (dates)**
- **TRUNC (dates)**

SAMPLE RELATION - PRODUCTS

P_ID	P_NAME	P_COST	MFG_DATE
122	Pepsodent	36.23	23-FEB-2017
124	Colgate	32.56	16-JUL-2017
132	Meswak		26-JAN-2018
145	Himalaya	56.58	30-SEP-2017
214	Dabur		24-APR-2017

DATE FUNCTIONS

SYSDATE

Returns the current system date and time on your local database.

Syntax:

SYSDATE

```
SQL> select SYSDATE from dual;

SYSDATE
---------
15-AUG-18
```

DATE FUNCTIONS

CURRENT_DATE

Returns the current date in the time zone of the current session.

Syntax:

CURRENT_DATE

```
SQL> select CURRENT_DATE from dual;

CURRENT_D
---------
15-AUG-18
```

DATE FUNCTIONS

SYSTIMESTAMP

Returns the current date and time.

It returns a TIMESTAMP WITH TIME ZONE value.

Syntax:

SYSTIMESTAMP

```
SQL> select SYSTIMESTAMP from dual;

SYSTIMESTAMP
---------------------------------------
15-AUG-18 04.51.42.428000 PM +05:30
```

DATE FUNCTIONS

CURRENT_TIMESTAMP

Returns the current date and time.

It returns a TIMESTAMP WITH TIME ZONE value.

Syntax:

CURRENT_TIMESTAMP

```
SQL> select CURRENT_TIMESTAMP from dual;

CURRENT_TIMESTAMP
---------------------------------------------------------------------------
15-AUG-18 04.53.03.205000 PM +05:30
```

DATE FUNCTIONS

LOCALTIMESTAMP

Returns the current date and time.

It returns a TIMESTAMP **WITHOUT TIME ZONE value.**

Syntax:

LOCALTIMESTAMP

```
SQL> select LOCALTIMESTAMP from dual;

LOCALTIMESTAMP
---------------------------------------------------------------------------
15-AUG-18 04.55.15.680000 PM
```

DATE FUNCTIONS

ADD_MONTHS

Returns a date with a specified number of months added.

Syntax:

ADD_MONTHS(Input date, number_of_months)

```
SQL> SELECT ADD_MONTHS('5-JUN-2018',5)FROM dual;

ADD_MONTH
---------
05-NOV-18

SQL> SELECT ADD_MONTHS('5-JUN-2018',12)FROM dual;

ADD_MONTH
---------
05-JUN-19
```

DATE FUNCTIONS

ADD_MONTHS

```
SQL> SELECT ADD_MONTHS(SYSDATE,12)FROM dual;

ADD_MONTH
---------
09-AUG-19

SQL> SELECT ADD_MONTHS(SYSDATE,-2)FROM dual;

ADD_MONTH
---------
09-JUN-18

SQL> SELECT ADD_MONTHS(SYSDATE,-12)FROM dual;

ADD_MONTH
---------
09-AUG-17
```

DATE FUNCTIONS

ADD_MONTHS

Display the product name, mfg_date and **add two months with mfg_date**

```
SQL> select p_name,mfg_date,ADD_MONTHS(mfg_date,2) from product;

P_NAME           MFG_DATE  ADD_MONTH
---------------- --------- ---------
Pepsodent        23-FEB-17 23-APR-17
Colgate          16-JUL-17 16-SEP-17
Meswak           26-JAN-18 26-MAR-18
Himalaya         30-SEP-17 30-NOV-17
Dabur            24-APR-17 24-JUN-17
```

DATE FUNCTIONS

MONTHS_BETWEEN

Returns the **number of months** between input date1 and input date2.

Syntax:

MONTHS_BETWEEN(input_date1,input_date2)

```
SQL> SELECT MONTHS_BETWEEN('12-JUL-2018','06-MAY-2018')FROM dual;

MONTHS_BETWEEN('12-JUL-2018','06-MAY-2018')
-------------------------------------------
                                 2.19354839
```

DATE FUNCTIONS

MONTHS_BETWEEN

```
SQL> SELECT MONTHS_BETWEEN('12-JUL-2018','06-MAY-2017')FROM dual;

MONTHS_BETWEEN('12-JUL-2018','06-MAY-2017')
-------------------------------------------
                                 14.1935484

SQL> SELECT ROUND(MONTHS_BETWEEN('12-JUL-2018','06-MAY-2017'))FROM dual;

ROUND(MONTHS_BETWEEN('12-JUL-2018','06-MAY-2017'))
--------------------------------------------------
                                                14
```

```
SQL> SELECT MONTHS_BETWEEN(SYSDATE,'01-JAN-2018')
  2  FROM dual;

MONTHS_BETWEEN(SYSDATE,'01-JAN-2018')
-------------------------------------
                             7.273523
```

DATE FUNCTIONS

MONTHS_BETWEEN

Find number of months between today's date and mfg_date.

```
SQL> select p_name,mfg_date,MONTHS_BETWEEN(SYSDATE,mfg_date) from product;

P_NAME          MFG_DATE  MONTHS_BETWEEN(SYSDATE,MFG_DATE)
--------------- --------- ---------------------------------
Pepsodent       23-FEB-17                        17.7648884
Colgate         16-JUL-17                        12.9906948
Meswak          26-JAN-18                        6.66811417
Himalaya        30-SEP-17                        10.5390819
Dabur           24-APR-17                        15.7326303

SQL> select p_name,mfg_date,ROUND(MONTHS_BETWEEN(SYSDATE,mfg_date)) from product
;

P_NAME          MFG_DATE  ROUND(MONTHS_BETWEEN(SYSDATE,MFG_DATE))
--------------- --------- ---------------------------------------
Pepsodent       23-FEB-17                                      18
Colgate         16-JUL-17                                      13
Meswak          26-JAN-18                                       7
Himalaya        30-SEP-17                                      11
Dabur           24-APR-17                                      16
```

DATE FUNCTIONS

LAST_DAY

Returns the last day of the month of the given date

Syntax:

LAST_DAY(input date)

Query to display last day of this month.

```
SQL> SELECT LAST_DAY<SYSDATE>FROM dual;

LAST_DAY<
----------
31-AUG-18
```

DATE FUNCTIONS

LAST_DAY

If you want **to find the first day of the next month, simply add one to the last_day results.**

```
SQL> select LAST_DAY(SYSDATE)+1 from dual;

LAST_DAY(
---------
01-SEP-18
```

Display the last day of the mfg_date of each Product.

```
SQL> select mfg_date,LAST_DAY(mfg_date) from product;

MFG_DATE  LAST_DAY(
--------- ---------
23-FEB-17 28-FEB-17
16-JUL-17 31-JUL-17
26-JAN-18 31-JAN-18
30-SEP-17 30-SEP-17
24-APR-17 30-APR-17
```

DATE FUNCTIONS

NEXT_DAY

Returns the first weekday that is greater than a *date*.

Syntax:

NEXT_DAY(date, weekday)

Weekday can be one of the following values

WEEKDAY	DESCRIPTION
SUNDAY	First Sunday greater than date
MONDAY	First Monday greater than date
TUESDAY	First Tuesday greater than date
WEDNESDAY	First Wednesday greater than date
THURSDAY	First Thursday greater than date
FRIDAY	First Friday greater than date
SATURDAY	First Saturday greater than date

DATE FUNCTIONS

NEXT_DAY

```
SQL> SELECT NEXT_DAY('10-AUG-2018','FRIDAY')FROM DUAL;

NEXT_DAY(
---------
17-AUG-18

SQL> SELECT NEXT_DAY('10-AUG-2018','MONDAY')FROM DUAL;

NEXT_DAY(
---------
13-AUG-18

SQL> SELECT NEXT_DAY('13-AUG-2018','MONDAY')FROM DUAL;

NEXT_DAY(
---------
20-AUG-18

SQL> SELECT NEXT_DAY('10-AUG-2018','SUNDAY')FROM DUAL;

NEXT_DAY(
---------
12-AUG-18
```

DATE FUNCTIONS

NEXT_DAY

Display the next Tuesday of the mfg_date of each Product.

```
SQL> select NEXT_DAY(mfg_date,'TUESDAY') from product;

NEXT_DAY(
---------
28-FEB-17
18-JUL-17
30-JAN-18
03-OCT-17
25-APR-17
```

DATE FUNCTIONS

EXTRACT

Extracts a **value** from a date.

Note:

- You can only extract YEAR, MONTH, and DAY from a DATE.
- You can only extract TIMEZONE_HOUR and TIMEZONE_MINUTE from a timestamp with a time zone datatype.

Syntax:

EXTRACT(YEAR/MONTH/DAY FROM input_date)

DATE FUNCTIONS

EXTRACT

To EXTRACT the YEAR

```
SQL> select EXTRACT(YEAR FROM mfg_date) from product;

EXTRACT(YEARFROMMFG_DATE)
-------------------------
                     2017
                     2017
                     2018
                     2017
                     2017
```

To EXTRACT the MONTH

```
SQL> select EXTRACT(MONTH FROM mfg_date) from product;

EXTRACT(MONTHFROMMFG_DATE)
--------------------------
                         2
                         7
                         1
                         9
                         4
```

EXTRACT function returns the month in NUMBER.

DATE FUNCTIONS

EXTRACT

To EXTRACT the DAY

```
SQL> select EXTRACT(DAY FROM mfg_date) from product;

EXTRACT(DAYFROMMFG_DATE)
------------------------
                      23
                      16
                      26
                      30
                      24
```

Display the details of the product which is manufactured in the year 2017.

```
SQL> select *from Product where EXTRACT(YEAR FROM mfg_date)=2017;

      P_ID P_NAME              P_COST MFG_DATE
---------- ---------- ---------- ---------
       122 Pepsodent          36.23 23-FEB-17
       124 Colgate            32.56 16-JUL-17
       145 Himalaya           56.58 30-SEP-17
       214 Dabur                    24-APR-17
```

DATE FUNCTIONS

EXTRACT

Display the details of the product which is manufactured in the month of February.

```
SQL> select *from Product where EXTRACT(MONTH FROM mfg_date)=2;

      P_ID P_NAME              P_COST MFG_DATE
---------- -------------- ---------- ---------
       122 Pepsodent           36.23 23-FEB-17
```

Display the details of the product which is manufactured before 20th of each month.

```
SQL> select *from Product where EXTRACT(DAY FROM mfg_date)<20;

      P_ID P_NAME              P_COST MFG_DATE
---------- -------------- ---------- ---------
       124 Colgate             32.56 16-JUL-17
```

DATE FUNCTIONS

EXTRACT

Display the details of product which is manufactured from jan to may, 2017.

```
SQL> select *from Product where EXTRACT(YEAR FROM mfg_date)=2017
  2  AND EXTRACT(MONTH FROM mfg_date) BETWEEN 1 AND 5;

      P_ID P_NAME              P_COST MFG_DATE
---------- --------------- ---------- ---------
       122 Pepsodent            36.23 23-FEB-17
       214 Dabur                      24-APR-17
```

DATE FUNCTIONS

ROUND

Returns a date rounded to a specific unit of measure.

Syntax:

ROUND(date [, format])

Round(date, 'year') – If the month is between Jan- Jun, the year will rounded to the beginning date of the current year and If the month is between July- Dec, the year will rounded to the beginning date of the next year.

```
SQL> select ROUND(mfg_date,'Year') from Product;

ROUND(MFG
---------
01-JAN-17
01-JAN-18
01-JAN-18
01-JAN-18
01-JAN-17
```

DATE FUNCTIONS

ROUND

Round(date, ‘month’) – If the date is between 1st – 15th, the month will rounded to the beginning date of the current month and If the date is between 16th – 31st, the month will rounded to the beginning date of the next month.

```
SQL> select ROUND(mfg_date,'Month') from Product;

ROUND(MFG
---------
01-MAR-17
01-AUG-17
01-FEB-18
01-OCT-17
01-MAY-17
```

DATE FUNCTIONS

ROUND

Round(date, ‘day’) – If the day is between Sunday - Wednesday, the date will rounded to the beginning date of the current week and If the day is between Thursday - Friday, the date will rounded to the beginning date of the next week.

```
SQL> select ROUND(mfg_date,'day') from Product;

ROUND(MFG
---------
26-FEB-17
16-JUL-17
28-JAN-18
01-OCT-17
23-APR-17
```

DATE FUNCTIONS

ROUND

Round(date, ‘Q’)

Quarter:

An year is divided into four quarters.

Quarter1 – January to March

Quarter2 – April to June

Quarter3 – July to September

Quarter4 – October to December

DATE FUNCTIONS

ROUND

Round(date, 'Q') – If the date is in the first half of the quarter (i.e., 1st Jan to 15th Feb or 1st April to 15th May or 1st July to 15th Aug or 1st Oct to 15th Nov), then the date will rounded to the beginning date of the current quarter and If the date is in the second half (i.e., 16th Feb to 31st Mar or 16th May to 31st June or 16th Aug to 31st Sep or 16th Nov to 31st Dec), then the date will rounded to the beginning date of the next quarter.

```
SQL> select mfg_date,ROUND(mfg_date,'Q') from Product;

MFG_DATE  ROUND(MFG
--------- ---------
23-FEB-17 01-APR-17
16-JUL-17 01-JUL-17
26-JAN-18 01-JAN-18
30-SEP-17 01-OCT-17
24-APR-17 01-APR-17
```

DATE FUNCTIONS

TRUNC

Returns a date truncated to a specific unit of measure.

Syntax:

TRUNC(date [, format])

Trunc(date, ‘year’) – The date will be truncated to the beginning of the current year.

```
SQL> select mfg_date,Trunc(mfg_date,'Year') from Product;

MFG_DATE  TRUNC(MFG
--------- ---------
23-FEB-17 01-JAN-17
16-JUL-17 01-JAN-17
26-JAN-18 01-JAN-18
30-SEP-17 01-JAN-17
24-APR-17 01-JAN-17
```

DATE FUNCTIONS

TRUNC

Trunc(date, ‘month’) – The date will be truncated to the beginning of the current month.

```
SQL> select mfg_date,Trunc(mfg_date,'Month') from Product;

MFG_DATE  TRUNC(MFG
--------- ---------
23-FEB-17 01-FEB-17
16-JUL-17 01-JUL-17
26-JAN-18 01-JAN-18
30-SEP-17 01-SEP-17
24-APR-17 01-APR-17
```

Trunc(date, ‘day’) – The date will be truncated to the beginning of the current week.

```
SQL> select mfg_date,Trunc(mfg_date,'Day') from Product;

MFG_DATE  TRUNC(MFG
--------- ---------
23-FEB-17 19-FEB-17
16-JUL-17 16-JUL-17
26-JAN-18 21-JAN-18
30-SEP-17 24-SEP-17
24-APR-17 23-APR-17
```

DATE FUNCTIONS

TRUNC

Trunc(date, ‘Q’) – The date will be truncated to the beginning of the current Quarter

```
SQL> select mfg_date,Trunc(mfg_date,'Q') from Product;

MFG_DATE  TRUNC(MFG
--------- ---------
23-FEB-17 01-JAN-17
16-JUL-17 01-JUL-17
26-JAN-18 01-JAN-18
30-SEP-17 01-JUL-17
24-APR-17 01-APR-17
```

CONVERSION FUNCTIONS

These are the functions that help us to convert a value in one form to another form.

- **TO_CHAR**

  converts a number or date to a string.

- **TO_DATE**

  converts a string to a date.

CONVERSION FUNCTIONS

Parameter	Explanation
YEAR	Year, spelled out
YYYY	4-digit year
YYY YY Y	Last 3, 2, or 1 digit(s) of year.
Q	Quarter of year (1, 2, 3, 4; JAN-MAR = 1).
MM	Month (01-12; JAN = 01).
MON	Abbreviated name of month.
MONTH	Name of month, padded with blanks to length of 9 characters.
WW	Week of year (1-53) where week 1 starts on the first day of the year and continues to the seventh day of the year.
W	Week of month (1-5) where week 1 starts on the first day of the month and ends on the seventh.

CONVERSION FUNCTIONS

Parameter	Explanation	Parameter	Explanation
D	Day of week (1-7).	HH24	Hour of day (0-23).
DAY	Name of day.	MI	Minute (0-59).
DD	Day of month (1-31).	SS	Second (0-59).
DDD	Day of year (1-366).	SSSSS	Seconds past midnight (0-86399).
DY	Abbreviated name of day.	AM, A.M., PM, or P.M.	Meridian indicator
HH	Hour of day (1-12).	AD or A.D	AD indicator
HH12	Hour of day (1-12).	BC or B.C.	BC indicator

CONVERSION FUNCTIONS

```
SQL> SELECT *FROM PRODUCT;

      P_ID P_NAME               P_COST MFG_DATE
---------- ------------------ ---------- ---------
       122 Pepsodent               36.23 23-FEB-17
       124 Colgate                 32.56 16-JUL-17
       132 Meswak                        26-JAN-18
       145 Himalaya                56.58 30-SEP-17
       214 Dabur                         24-APR-17
```

TO GET YEAR FROM DATE COLUMN [Year, spelled out]:

```
SQL> SELECT TO_CHAR(MFG_DATE,'YEAR') FROM PRODUCT;

TO_CHAR(MFG_DATE,'YEAR')
----------------------------------------
TWENTY SEVENTEEN
TWENTY SEVENTEEN
TWENTY EIGHTEEN
TWENTY SEVENTEEN
TWENTY SEVENTEEN
```

CONVERSION FUNCTIONS

[4-digit year]:

```
SQL> SELECT TO_CHAR(MFG_DATE,'YYYY') FROM PRODUCT;

TO_C
----
2017
2017
2018
2017
2017
```

[3-digit year]:

```
SQL> SELECT TO_CHAR(MFG_DATE,'YYY') FROM PRODUCT;

TO_
---
017
017
018
017
017
```

CONVERSION FUNCTIONS

[2-digit year]:

```
SQL> SELECT TO_CHAR(MFG_DATE,'YY') FROM PRODUCT;

TO
--
17
17
18
17
17
```

[1-digit year]:

```
SQL> SELECT TO_CHAR(MFG_DATE,'Y') FROM PRODUCT;

T
-
7
7
8
7
7
```

CONVERSION FUNCTIONS

TO GET QUARTER OF YEAR (1, 2, 3, 4; JAN-MAR = 1)

```
SQL> SELECT TO_CHAR(MFG_DATE,'Q') FROM PRODUCT;

T
-
1
3
1
3
2
```

TO GET MONTH OF YEAR [MONTH - Name of month]

```
SQL> SELECT TO_CHAR(MFG_DATE,'MONTH') FROM PRODUCT;

TO_CHAR(MFG_DATE,'MONTH')
-------------------------
FEBRUARY
JULY
JANUARY
SEPTEMBER
APRIL
```

CONVERSION FUNCTIONS

[MON - Abbreviated name of month]

```
SQL> SELECT TO_CHAR(MFG_DATE,'MON') FROM PRODUCT;

TO_CHAR(MFG_
------------
FEB
JUL
JAN
SEP
APR
```

[MM - Month (01-12; JAN = 01 AND SO ON)]

```
SQL> SELECT TO_CHAR(MFG_DATE,'MM') FROM PRODUCT;

TO
--
02
07
01
09
04
```

CONVERSION FUNCTIONS

[DAY - Name of day]

```
SQL> SELECT TO_CHAR(MFG_DATE,'DAY') FROM PRODUCT;

TO_CHAR(MFG_DATE,'DAY')
------------------------------------
THURSDAY
SUNDAY
FRIDAY
SATURDAY
MONDAY
```

[DY - Abbreviated name of day]

```
SQL> SELECT TO_CHAR(MFG_DATE,'DY') FROM PRODUCT;

TO_CHAR(MFG_
------------
THU
SUN
FRI
SAT
MON
```

CONVERSION FUNCTIONS

[DDD - Day of year (1-366)]

```
SQL> SELECT TO_CHAR(MFG_DATE,'DDD') FROM PRODUCT;

TO_
---
054
197
026
273
114
```

[DD - Day of month (1-31)]

```
SQL> SELECT TO_CHAR(MFG_DATE,'DD') FROM PRODUCT;

TO
--
23
16
26
30
24
```

CONVERSION FUNCTIONS

[Week of month (1-5)]

```
SQL> SELECT TO_CHAR(MFG_DATE,'W') FROM PRODUCT;

T
-
4
3
4
5
4
```

[Week of year (1-53)]

```
SQL> SELECT TO_CHAR(MFG_DATE,'WW') FROM PRODUCT;

TO
--
08
29
04
39
17
```

CONVERSION FUNCTIONS

TO DISPLAY DATE IN THE [15TH August 2018] FORMAT:

```
SQL> SELECT TO_CHAR(MFG_DATE,'DDth Month YYYY') FROM PRODUCT;

TO_CHAR(MFG_DATE,'DDTHMONTHYYYY')
---------------------------------------
23RD February  2017
16TH July      2017
26TH January   2018
30TH September 2017
24TH April     2017
```

TO DISPLAY DATE IN THE [August 15TH 2018] FORMAT:

```
SQL> SELECT TO_CHAR(MFG_DATE,'Month DDth YYYY') FROM PRODUCT;

TO_CHAR(MFG_DATE,'MONTHDDTHYYYY')
---------------------------------------
February  23RD 2017
July      16TH 2017
January   26TH 2018
September 30TH 2017
April     24TH 2017
```

CONVERSION FUNCTIONS

TO ADD ‘$’ BEFORE P_COST:

```
SQL> SELECT TO_CHAR(P_COST,'$999.99') FROM PRODUCT;

TO_CHAR(
--------
  $36.23
  $32.56

  $56.58
```

TO DISPLAY P_COST AS A WHOLE NUMBER:

```
SQL> SELECT TO_CHAR(P_COST,'00099') FROM PRODUCT;

TO_CHA
------
 00036
 00033

 00057
```

CONVERSION FUNCTIONS

TO_DATE:

To Convert a string into date.

```
SQL> SELECT TO_DATE('21/03/2013','DD/MM/YYYY') FROM DUAL;

TO_DATE('
---------
21-MAR-13

SQL> SELECT TO_DATE('21032013','DDMMYYYY') FROM DUAL;

TO_DATE('
---------
21-MAR-13
```

CONVERSION FUNCTIONS

```
SQL> SELECT TO_DATE('131113','YYMMDD') FROM DUAL;

TO_DATE('
---------
13-NOV-13

SQL> SELECT TO_DATE('20131113','YYYYMMDD') FROM DUAL;

TO_DATE('
---------
13-NOV-13
```

AGGREGATE FUNCTIONS

AGGREGATE FUNCTIONS

- These functions perform a summary operation on all the values that a query returns. Also called as multi-row functions.
- These functions don't handle NULL in the same way as ordinary functions and operators. Instead of returning NULL as soon as a NULL operand is encountered, they only take non-NULL fields into consideration while computing the outcome.
- Another thing worth knowing is that COUNT(*) and COUNT(FieldName) never return NULL: if there are no rows in the set, both functions return 0. COUNT(FieldName) also returns 0 if all FieldName fields in the set are NULL. The other aggregate functions return NULL in such cases.

- **MIN**
- **MAX**
- **SUM**
- **AVG**
- **COUNT**

AGGREGATE FUNCTIONS

MAX

Returns the maximum value of an expression.

Syntax:

select MAX(Column_Name) from table_name [where conditions];

```
SQL> select MAX(P_Cost) from Product;

MAX(P_COST)
-----------
      56.58

SQL> select MAX(P_Cost) from Product where P_Name LIKE 'P%';

MAX(P_COST)
-----------
      36.23
```

AGGREGATE FUNCTIONS

MIN

Returns the minimum value of an expression.

Syntax:

select MIN(Column_Name) from table_name [where conditions];

```
SQL> select MIN(P_Cost) from Product;

MIN(P_COST)
-----------
      32.56

SQL> select MIN(P_Cost) from Product where P_ID < 130;

MIN(P_COST)
-----------
      32.56
```

AGGREGATE FUNCTIONS

SUM

Returns the summed value of an expression.

Syntax:

select SUM(Column_Name) from table_name [where conditions];

```
SQL> select SUM(P_Cost) from Product;

SUM(P_COST)
-----------
     125.37

SQL> select SUM(P_Cost) from Product where P_ID > 130;

SUM(P_COST)
-----------
      56.58
```

AGGREGATE FUNCTIONS

AVG

Returns the average value of an expression.

Syntax:

select AVG(Column_Name) from table_name [where conditions];

```
SQL> select AVG(P_Cost) from Product;

AVG(P_COST)
-----------
      41.79

SQL> select AVG(P_Cost) from Product where P_ID > 130;

AVG(P_COST)
-----------
      56.58
```

AGGREGATE FUNCTIONS

COUNT

Returns the count of an expression.

Syntax:

select COUNT(Column_Name) from table_name [where conditions];

```
SQL> select COUNT(P_Cost) from Product;

COUNT(P_COST)
-------------
            3

SQL> select COUNT(P_Name) from Product where P_Cost IS Null;

COUNT(P_NAME)
-------------
            2
```

AGGREGATE FUNCTIONS

COUNT

Query to display number of records in products relation.

```
SQL> select COUNT(*) from Product;

  COUNT(*)
----------
         5
```

Note: The COUNT function will only include the records in the count where the value of expression in COUNT(expression) is NOT NULL. When expression contains a NULL value, it is not included in the COUNT calculations whereas count(*) includes NULL values also.