

USING CONVERSION FUNCTIONS

http://www.tutorialspoint.com/sql_certificate/conversion_functions.htm

Copyright © tutorialspoint.com

Besides the SQL utility functions, Oracle inbuilt function library contains type conversion functions. There may be scenarios where the query expects input in a specific data type, but it receives it in a different data type. In such cases, Oracle implicitly tries to convert the unexpected value to a compatible data type which can be substituted in place and application continuity is not compromised. Type conversion can be either implicitly done by Oracle or explicitly done by the programmer.

Implicit data type conversion works based on a matrix which showcases the Oracle's support for internal type casting. Besides these rules, Oracle offers type conversion functions which can be used in the queries for explicit conversion and formatting. As a matter of fact, it is recommended to perform explicit conversion instead of relying on software intelligence. Though implicit conversion works well, but to eliminate the skew chances where bad inputs could be difficult to typecast internally.

Implicit Data Type Conversion

A VARCHAR2 or CHAR value can be implicitly converted to NUMBER or DATE type value by Oracle. Similarly, a NUMBER or DATE type value can be automatically converted to character data by Oracle server. Note that the implicit interconversion happens only when the character represents the a valid number or date type value respectively.

For example, examine the below SELECT queries. Both the queries will give the same result because Oracle internally treats 15000 and '15000' as same.

Query-1

```
SELECT employee_id, first_name, salary
FROM employees
WHERE salary > 15000;
```

Query-2

```
SELECT employee_id, first_name, salary
FROM employees
WHERE salary > '15000';
```

Explicit Data Type Conversion

SQL Conversion functions are single row functions which are capable of typecasting column value, literal or an expression . TO_CHAR, TO_NUMBER and TO_DATE are the three functions which perform cross modification of data types.

TO_CHAR function

TO_CHAR function is used to typecast a numeric or date input to character type with a format model *optional*.

Syntax

```
TO_CHAR(number1, [format], [nls_parameter])
```

For number to character conversion, nls parameters can be used to specify decimal characters, group separator, local currency model, or international currency model. It is an optional specification - if not available, session level nls settings will be used. For date to character conversion, the nls parameter can be used to specify the day and month names, as applicable.

Dates can be formatted in multiple formats after converting to character types using TO_CHAR function. The TO_CHAR function is used to have Oracle 11g display dates in a particular format.

Format models are case sensitive and must be enclosed within single quotes.

Consider the below SELECT query. The query format the HIRE_DATE and SALARY columns of EMPLOYEES table using TO_CHAR function.

```
SELECT first_name,
       TO_CHAR (hire_date, 'MONTH DD, YYYY') HIRE_DATE,
       TO_CHAR (salary, '$99999.99') Salary
FROM employees
WHERE rownum < 5;
```

FIRST_NAME	HIRE_DATE	SALARY
Steven	JUNE 17, 2003	\$24000.00
Neena	SEPTEMBER 21, 2005	\$17000.00
Lex	JANUARY 13, 2001	\$17000.00
Alexander	JANUARY 03, 2006	\$9000.00

The first TO_CHAR is used to convert the hire date to the date format MONTH DD, YYYY i.e. month spelled out and padded with spaces, followed by the two-digit day of the month, and then the four-digit year. If you prefer displaying the month name in mixed case *that is*, " December " , simply use this case in the format argument: 'MonthDD, YYYY'.

The second TO_CHAR function in Figure 10-39 is used to format the SALARY to display the currency sign and two decimal positions.

Oracle offers comprehensive set of format models. The below table shows the list of format models which can be used to typecast date and number values as character using TO_CHAR.

Format Model	Description
,comma	It returns a comma in the specified position. You can specify multiple commas in a number format model. Restrictions:A comma element cannot begin a number format model. A comma cannot appear to the right of a decimal character or period in a number format model.
.period	Returns a decimal point, which is a period . in the specified position. Restriction: You can specify only one period in a number format model
\$	Returns value with a leading dollar sign
0	Returns leading zeros. Returns trailing zeros.
9	Returns value with the specified number of digits with a leading space if positive or with a leading minus if negative. Leading zeros are blank, except for a zero value, which returns a zero for the integer part of the fixed-point number.
B	Returns blanks for the integer part of a fixed-point number when the integer part is zero <i>regardless of " 0 " in the format model.</i>
C	Returns in the specified position the ISO currency symbol <i>the current value of the NLS_ISO_CURRENCY parameter.</i>
D	Returns in the specified position the decimal character, which is the current value of the NLS_NUMERIC_CHARACTER parameter. The default is a period . . Restriction: You can specify only one decimal character in a number format model.
EEE	Returns a value using in scientific notation.
FM	Returns a value with no leading or trailing blanks.
G	Returns in the specified position the group separator <i>the current value of the NLS_NUMERIC_CHARACTER parameter.</i> You can specify multiple

	group separators in a number format model. Restriction: A group separator cannot appear to the right of a decimal character or period in a number format model
L	Returns in the specified position the local currency symbol <i>thecurrentvalueoftheNLS_CURRENCYparameter.</i>
MI	Returns negative value with a trailing minus sign -. Returns positive value with a trailing blank. Restriction: The MI format element can appear only in the last position of a number format model.
PR	Returns negative value in . It can appear only in the end of a number format model.
RN,rm	Returns a value as Roman numerals in uppercase. Returns a value as Roman numerals in lowercase. Value can be an integer between 1 and 3999.
S	Returns negative value with a leading or trailing minus sign -. Returns positive value with a leading or trailing plus sign +. Restriction: The S format element can appear only in the first or last position of a number format model.
TM	"Text minimum". Returns <i>indecimaloutput</i> the smallest number of characters possible. This element is case-insensitive.
U	Returns in the specified position the "Euro" or other dual currency symbol <i>thecurrentvalueoftheNLS_DUAL_CURRENCYparameter.</i>
V	Returns a value multiplied by 10 ⁿ <i>andifnecessary, rounditup</i> , where n is the number of 9's after the "V".
X	Returns the hexadecimal value of the specified number of digits.

TO_NUMBER function

The TO_NUMBER function converts a character value to a numeric datatype. If the string being converted contains nonnumeric characters, the function returns an error.

Syntax

```
TO_NUMBER (string1, [format], [nls_parameter])
```

The below table shows the list of format models which can be used to typecast character values as number using TO_NUMBER.

Format Model	Description
CC	Century
SCC	Century BC prefixed with -
YYYY	Year with 4 numbers
SYYY	Year BC prefixed with -
IYYY	ISO Year with 4 numbers
YY	Year with 2 numbers
RR	Year with 2 numbers with Y2k compatibility
YEAR	Year in characters

SYEAR	Year in characters, BC prefixed with -
BC	BC/AD Indicator
Q	Quarter in numbers 1, 2, 3, 4
MM	Month of year 01, 02...12
MONTH	Month in characters <i>i. e. January</i>
MON	JAN, FEB
WW	Week number <i>i. e. 1</i>
W	Week number of the month <i>i. e. 5</i>
IW	Week number of the year in ISO standard.
DDD	Day of year in numbers <i>i. e. 365</i>
DD	Day of the month in numbers <i>i. e. 28</i>
D	Day of week in numbers <i>i. e. 7</i>
DAY	Day of the week in characters <i>i. e. Monday</i>
FMDAY	Day of the week in characters <i>i. e. Monday</i>
DY	Day of the week in short character description <i>i. e. SUN</i>
J	Julian Day <i>number of days since January 14713BC, where January 14713BC is 1 in Oracle</i>
HH,H12	Hour number of the day 1 – 12
HH24	Hour number of the day with 24Hours notation 0 – 23
AM, PM	AM or PM
MI, SS	Number of minutes and seconds <i>i. e. 59 ,</i>
SSSSS	Number of seconds this day.
DS	Short date format. Depends on NLS-settings. Use only with timestamp.
DL	Long date format. Depends on NLS-settings. Use only with timestamp.
E	Abbreviated era name. Valid only for calendars: Japanese Imperial, ROC Official, Thai Buddha.
EE	The full era name
FF	The fractional seconds. Use with timestamp.
FF1..FF9	The fractional seconds. Use with timestamp. The digit controls the number of decimal digits used for fractional seconds.
FM	Fill Mode: suppresses blanks in output from conversion
FX	Format Exact: requires exact pattern matching between data and format model.
IYY OR IY OR I	The last 3,2,1 digits of the ISO standard year. Output only
RM	The Roman numeral representation of the month <i>I. . XII</i>
RR	The last 2 digits of the year.
RRRR	The last 2 digits of the year when used for output. Accepts four-digit years when used for input.

SP	Spelled format. Can appear of the end of a number element. The result is always in english. For example month 10 in format MMSP returns "ten"
SPTH	Spelled and ordinal format; 1 results in first.
TH	Converts a number to it's ordinal format. For example 1 becoms 1st.
TS	Short time format. Depends on NLS-settings. Use only with timestamp.
TZD	Abbreviated time zone name. ie PST.
TZH,TZM	Time zone hour/minute displacement.
TZR	Time zone region
X	Local radix character. In America this is a period .

The SELECT queries below accept numbers as character inputs and prints them following the format specifier.

```
SELECT TO_NUMBER('121.23', '9G999D99')
FROM DUAL
```

```
TO_NUMBER('121.23', '9G999D99')
-----
                        121.23
```

```
SELECT TO_NUMBER('1210.73', '9999.99')
FROM DUAL;
```

```
TO_NUMBER('1210.73', '9999.99')
-----
                        1210.73
```

TO_DATE function

The function takes character values as input and returns formatted date equivalent of the same. The TO_DATE function allows users to enter a date in any format, and then it converts the entry into the default format used by Oracle 11g.

Syntax:

```
TO_DATE( string1, [ format_mask ], [ nls_language ] )
```

A format_mask argument consists of a series of elements representing exactly what the data should look like and must be entered in single quotation marks.

Format Model	Description
YEAR	Year, spelled out
YYYY	4-digit year
YYY,YY,Y	Last 3, 2, or 1 digits of year.
IYY,IY,I	Last 3, 2, or 1 digits of ISO year.
IYYY	4-digit year based on the ISO standard
RRRR	Accepts a 2-digit year and returns a 4-digit 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.
RM	Roman numeral month I – XII; JAN = I.
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.
IW	Week of year 1 – 52or1 – 53 based on the ISO standard.
D	Day of week 1 – 7.
DAY	Name of day.
DD	Day of month 1 – 31.
DDD	Day of year 1 – 366.
DY	Abbreviated name of day.
J	Julian day; the number of days since January 1, 4712 BC.
HH12	Hour of day 1 – 12.
HH24	Hour of day 0 – 23.
MI,SS	Minute 0 – 59.
SSSSS	Seconds past midnight 0 – 86399.
FF	Fractional seconds. Use a value from 1 to 9 after FF to indicate the number of digits in the fractional seconds. For example, 'FF4'.
AM,PM	Meridian indicator
AD,BC	AD, BC indicator
TZD	Daylight savings information. For example, 'PST'
TZH,TZM,TZR	Time zone hour/minute/region.

The following example converts a character string into a date:

```
SELECT TO_DATE('January 15, 1989, 11:00 A.M.', 'Month dd, YYYY, HH:MI A.M.',
'NLS_DATE_LANGUAGE = American')
FROM DUAL;

TO_DATE('
-----
15-JAN-89
```

General Functions

General functions are used to handle NULL values in database. The objective of the general NULL handling functions is to replace the NULL values with an alternate value. We shall briefly see through these functions below.

NVL

The NVL function substitutes an alternate value for a NULL value.

Syntax:

```
NVL( Arg1, replace_with )
```

In the syntax, both the parameters are mandatory. Note that NVL function works with all types of data types. And also that the data type of original string and the replacement must be in compatible state i.e. either same or implicitly convertible by Oracle.

If arg1 is a character value, then oracle converts replacement string to the data type compatible with arg1 before comparing them and returns VARCHAR2 in the character set of expr1. If arg1 is numeric, then Oracle determines the argument with highest numeric precedence, implicitly converts the other argument to that data type, and returns that data type.

The SELECT statement below will display 'n/a' if an employee has been not assigned to any job yet i.e. JOB_ID is NULL. Otherwise, it would display the actual JOB_ID value.

```
SELECT first_name, NVL(JOB_ID, 'n/a')
FROM employees;
```

NVL2

As an enhancement over NVL, Oracle introduced a function to substitute value not only for NULL columns values but also for NOT NULL columns. NVL2 function can be used to substitute an alternate value for NULL as well as non NULL value.

Syntax:

```
NVL2( string1, value_if_NOT_null, value_if_null )
```

The SELECT statement below would display 'Bench' if the JOB_CODE for an employee is NULL. For a definite not null value of JOB CODE, it would show constant value 'Job Assigned'.

```
SQL> SELECT NVL2(JOB_CODE, 'Job Assigned', 'Bench')
FROM employees;
```

NULLIF

The NULLIF function compares two arguments expr1 and expr2. If expr1 and expr2 are equal, it returns NULL; else, it returns expr1. Unlike the other null handling function, first argument can't be NULL.

Syntax:

```
NULLIF (expr1, expr2)
```

Note that first argument can be an expression that evaluates to NULL, but it can't be the literal NULL. Both the parameters are mandatory for the function to execute.

The below query returns NULL since both the input values, 12 are equal.

```
SELECT NULLIF (12, 12)
FROM DUAL;
```

Similarly, below query return 'SUN' since both the strings are not equal.

```
SELECT NULLIF ('SUN', 'MOON')
FROM DUAL;
```

COALESCE

COALESCE function, a more generic form of NVL, returns the first non-null expression in the argument list. It takes minimum two mandatory parameters but maximum arguments has no limit.

Syntax:

```
COALESCE (expr1, expr2, ... expr_n )
```

Consider the below SELECT query. It selects the first not null value fed into address fields for an employee.

```
SELECT COALESCE (address1, address2, address3) Address  
FROM employees;
```

Interestingly, the working of COALESCE function is similar to IF..ELSIF..ENDIF construct. The query above can be re-written as -

```
IF address1 is not null THEN  
    result := address1;  
ELSIF address2 is not null THEN  
    result := address2;  
ELSIF address3 is not null THEN  
    result := address3;  
ELSE  
    result := null;  
END IF;
```

Conditional Functions

Oracle provides conditional functions DECODE and CASE to impose conditions even in SQL statement.

The DECODE function

The function is the SQL equivalence of IF..THEN..ELSE conditional procedural statement. DECODE works with values/columns/expressions of all data types.

Syntax:

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

DECODE function compares expression against each search value in order. If equality exists between expression and search argument, then it returns the corresponding result. In case of no match, default value is returned, if defined, else NULL. In case of any type compatibility mismatch, oracle internally does possible implicit conversion to return the results.

As a matter of fact, Oracle considers two nulls to be equivalent while working with DECODE function.

```
SELECT DECODE(NULL, NULL, 'EQUAL', 'NOT EQUAL')  
FROM DUAL;  
  
DECODE  
-----  
EQUAL
```

If expression is null, then Oracle returns the result of the first search that is also null. The maximum number of components in the DECODE function is 255.

```
SELECT first_name, salary, DECODE (hire_date, sysdate, 'NEW JOINEE', 'EMPLOYEE')
```

```
FROM employees;
```

CASE expression

CASE expressions work on the same concept as DECODE but differ in syntax and usage.

Syntax:

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

Oracle search starts from left and moves rightwards until it finds a true condition, and then returns result expression associated with it. If no condition is found to be true, and an ELSE clause exists, then Oracle returns result defined with else. Otherwise, Oracle returns null.

The maximum number of arguments in a CASE expression is 255. All expressions count toward this limit, including the initial expression of a simple CASE expression and the optional ELSE expression. Each WHEN ... THEN pair counts as two arguments. To avoid exceeding this limit, you can nest CASE expressions so that the return_expr itself is a CASE expression.

```
SELECT first_name, CASE WHEN salary < 200 THEN 'GRADE 1'  
  WHEN salary > 200 AND salary < 5000 THEN 'GRADE 2'  
  ELSE 'GRADE 3'  
  END CASE  
FROM employees;
```

ENAM	CASE
JOHN	GRADE 2
EDWIN	GRADE 3
KING	GRADE 1

Processing math: 100%