# Week 3: Single-Row Functions

**SQL Functions**

Functions are program that take zero or more argument and return a single value as output. A function makes a simple query more powerful and is used to manipulate data values.

Functions are one of the most powerful features of SQL and can be used to:

* Perform calculation on data
* Modify individual data items
* Manipulate output for groups of rows
* Format dates and numbers for display
* Convert column data types

SQL consists of a wide variety of functions to perform the above mentioned tasks and these functions can be called from SQL statements (most of the time its SELECT statement). There are five significant classes of functions these are:

1. Single-row functions
2. Aggregate/ Multiple-row functions (also known as Group function)
3. Analytical functions
4. Object-reference functions
5. Programmer defined functions

But for our understanding we will discuss only the first two functions which are discussed this week and in the weeks coming ahead.

**SQL Functions**

**Single Input**

**Multiple Inputs**

**Single-Row Function**

**Aggregate/ Multiple-Row Function**

**Single output**

**Single output**

**Single-Row Function:** These functions operate on single row only and return one result per row. There are different types of Single-row function which are categorised under the following heads (discussed in this week’s notes). These categories are:

* General
* Character
* Number
* Date
* Conversion

**Aggregate/Multiple-Row Function:** These functions can manipulate groups of rows to produce one result per group of rows. These functions are also known as Group functions (discussed in detail in week 5).

These SQL functions sometimes take an argument and always return a value. Also these functions can be nested, so that the output from one function can be used as input to another function. In the nested functions we can always see a combination of both single-row and group function, as nesting is not defined to only on type of function.

This week we will be focusing more on the Single-Row functions and will discuss about the Group function in detail in week 5.

**Single-Row Functions**

As we have read earlier that Single-Row function operate on expressions derived from a column or literals and they are executed once for the each row retrieved. They accept one or more arguments to perform an action to demonstrate the effect of function on the value provided. A simple syntax for any function is given below:

|  |
| --- |
| **Syntax:**  Function\_Name [(arg1, arg2 …)] |

In the syntax:

**Function\_Name** is the name of the SQL Function

**Arg1, arg2…** is the argument used by the function

An argument can be anything and of course one of the following:

* User supplied content
* Variable value
* Column name
* Expression

A function can use any of these as an argument and give a relevant output. Single-Row function have incorporated some feature which makes them distinct from others, these features are as follows:

* Acting on each row returned in the query
* Returning one result per row
* Possibly returning a data value of a different type than that referenced
* Possibly expecting one or more arguments
* Can be used in SELECT, WHERE and ORDER BY clause and can also be nested.

As mentioned earlier Single-Row functions are categorised under five heads; the figure below shows the various categories of function:

**Character**

**Number**

**General**

**Single-Row Functions**

**Data**

**Conversion**

Let us now discuss all these categories and the function coming under them in detail one by one.

**Character Functions**

Single-Row character function deals with the character data. Most of the functions falling under this category take one or more character arguments and mostly return character values, but there are some character functions which return numeric values. The character function is divided into two sub-categories, these are:

* **Case-manipulation Functions:** This sub-category of function converts the case of the character string. There are basically three functions which fall under this sub-category; these are:

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Purpose** | **Example** | **Result** |
| **LOWER**(*column |expression*) | It is used to convert a string into Lower case. | **SELECT LOWER(‘Sunny Holidays’)**  **FROM Dual;** | sunny holidays |
| **UPPER**(*column |expression*) | It is used to convert a string into Upper case. | **SELECT UPPER(‘Sunny holidays’)**  **FROM Dual;** | SUNNY HOLIDAYS |
| **INITCAP**(*column |expression*) | It returns the string with the first letter of each word in upper case. | **SELECT INITCAP(‘sunny holidays’) FROM Dual;** | Sunny Holidays |

|  |  |  |
| --- | --- | --- |
| **Example 3.1: Using Case-manipulation character function**   |  | | --- | | **SELECT UPPER(Fname)**  **FROM Author;** | |  | |
|  |

Example 3.1 shows the use of functions with column name as an argument. In this example we are using UPPER function which convert author’s first name to upper case.

* **Character- manipulation Functions:** This sub-category of character function is used to manipulate the characters within the string and as well as the strings itself. Various functions which falls under this category are given below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Purpose** | **Example** | **Result** |
| **CONCAT**(*column1|expression,*  *Column2|expression2*) | Concatenates two strings. It is same as the || operator | **SELECT CONCAT(‘Scientific’, ‘Research’)**  **FROM Dual;** | ScientificReserch |
| **SUBSTR**(*column |expression, m [,n]*) | This function returns *column |expression* which is *n* long, beginning at position *m*. If *m* is negative, the position is counted from right to left i.e. backward. | **SELECT SUBSTR(‘Tales of Two Cities’, 3,8) FROM Dual;** | Les of T |
| **LENGTH**(*column |expression*) | This function returns the numeric length in characters of *column |expression.* | **SELECT LENGTH(‘Tales of Two Cities’) FROM Dual;** | 19 |
| **INSTR**(*column |expression, ’string’, m [,n]*) | This function returns the numeric character position in *column |expression* where the *nth*occurrence of *string* is found. The search begins at the *mth* character position in *column |expression*, if its negative the search starts backward. | **SELECT INSTR(‘The Three Musketeers’, ‘K’) FROM Dual;** | 14 |
| **LPAD**(*column |expression, m, ‘string’*) | This function returns the *column| expression* expanded in length to *m* characters, using *string* to fill in spaces as needed on the left of *column| expression.* | **SELECT LPAD(‘HELLO’, 8, ’\*’) FROM Dual;** | \*\*\*HELLO |
| **RPAD**(*column |expression, m, ‘string’*) | This function returns the *column| expression* expanded in length to *m* characters, using *string* to fill in spaces as needed on the right of *column| expression.* | **SELECT RPAD (‘HELLO’, 8, ‘\*’) FROM Dual;** | HELLO\*\*\* |
| **TRIM**(*leading |trailing |both, trim\_character FROM trim\_source*) | This function returns the *trim\_source* with all *leading | trailing | both* occurrences of characters in *trim\_character* removed. A NULL is returned if any of the three arguments is NULL. | **SELECT TRIM(‘T’ FROM ‘The Three Musketeers’) FROM Dual;** | he Three Musketeers |
| **REPLACE**(*text, search\_string, replacement\_string*) | This function returns *text* with all the occurrences of *search\_string* replaced with *replacement\_string*. If replacement\_string is NULL then all the occurrences of search\_string are removed. | **SELECT REPLACE(‘Sunny Weekend’, ’Sunny’, ’Fun’) FROM Dual;** | Fun Weekend |

|  |  |  |
| --- | --- | --- |
| **Example 3.2: Using Character-manipulation character function**   |  | | --- | | **SELECT Author\_id, CONCAT(Fname, Lname) AS “Author Full Name”, LENGTH(CONCAT(Fname, Lname)) AS “Length of Full Name”**  **FROM Author;** | |  | |
|  |

Example 3.2 demonstrates the use of character-manipulation function is SELECT statement. It also show the use of nested Character functions. When we have a nested function at that point the inner most function is executed first and the result of that function is used as the input to function covering that function. Here in this example first the innermost function i.e. the CONCAT function is executed and the result of that function is provided to the LENGTH function to execute. In such a way we can use as many functions in the nesting.

Apart from the case-manipulation and character-manipulation functions there are few more functions which fall in the category of Character functions, these are as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Purpose** | **Example** | **Result** |
| **ASCII**(*String*) | This function returns the ASCII decimal equivalent of the first character in the *String* | **SELECT ASCII(‘Hello’) FROM Dual;** | 72 |
| **CHR**(*m [ USING NCHAR\_CS]*) | This function returns the character equivalent of the decimal (binary) representation of the character. If the optional USING NCHAR\_CS is included, the character from National character set is returned. | **SELECT CHR(78) FROM Dual;** | N |
| **LTRIM**(*column| expression, [,string]*) | This function returns the *column| expression* without any leading characters that appears in *string*. If no *string* characters are leading character in *column| expression* then it is returned unchanged. | **SELECT LTRIM(‘xyXYyx LTRIM WORD’, ‘xy’) AS “Effect of LTRIM” FROM Dual;** | XYyx LTRIM WORD |
| **RTRIM**(*character\_string1, [,character\_string2]*) | This function returns the *column| expression* without any trailing characters that appears in *string*. If no *string* characters are trailing character in *column| expression* then it is returned unchanged. | **SELECT RTRIM(‘xyXYyx RTRIM WORD xyXYyx’, ‘yx’) AS “Effect of RTRIM” FROM Dual;** | xyXY RTRIM WORD |

**Numeric Functions**

Single-Row numeric functions deals with the numeric data and perform some kind of mathematical or arithmetic manipulation. All the function coming under this category take some argument and returns a numeric values. There are trigonometric functions also which comes under this category but for the time we will only take few widely used functions into considerations. These functions are given below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Purpose** | **Example** | **Result** |
| **ABS**(*n*) | This function returns the absolute value of *n*. | **SELECT ABS(-23) AS “Negative Value” FROM Dual;** | NEGATIVE VALUE  -------------------------  23 |
| **CEIL**(*n*) | This function returns the smallest integer that is greater than or equals to *n*. it rounds up to a whole number. | **SELECT CEIL(-32.86) AS “Round up whole number” FROM Dual;** | Round up whole number  ---------------------------------  -32 |
| **FLOOR**(*n*) | This function returns the largest integer that is less than or equal to *n*. It rounds down to a whole number. | **SELECT FLOOR(-32.86) AS “Round down whole number” FROM Dual;** | Round down whole number  ---------------------------------  -33 |
| **MOD**(*n1, n2*) | This function returns *n1* modulo *n2* or the remainder of *n1* divided by *n2*. If the *n1* is negative the result is negative. The sign of *n2* doesn’t have any effect on the result. | **SELECT MOD(8,3) AS “8 divided by 3” FROM Dual;** | 8 divided by 3  -------------------------  2 |
| **ROUND**(*n, i*) | This function returns *n* rounded to *i* digits of precision to the right of the decimal. If *i* is negative, *i* is rounded to left of the decimal. | **SELECT ROUND(123.4523,2) AS “Positive round to right” FROM Dual;** | Positive round to right  ---------------------------------  123.45 |
| **TRUNC**(*n, i*) | This function returns *n* truncated to *i* digits of precision to the right of the decimal. If *i* is negative *n* is truncated to left of the decimal. | **SELECT TRUNC(49.926,2) FROM Dual;** | TRUNC(49.926)  ---------------------  49.92 |

These are the few Numeric functions which are being used. The Example 3.3 below shoe how to use the numeric function with a select statement.

|  |  |  |
| --- | --- | --- |
| **Example 3.3: Using Numeric function with SELECT statement**   |  | | --- | | **SELECT Title, Price, Paperback, ROUND(((Price\*10)/100), 2) AS “10 % discount”**  **FROM Books**  **WHERE Paperback LIKE ‘t’;** | |  | |
|  |

Here in this example we are using the numeric function as well as the arithmetic operator to demonstrate the use of function with operators.

**Date Functions**

Single-row date function deals with the data of data type *DATE*. Oracle database stores the date in a numeric format, representing the country, year, month, day, hours, minutes and seconds.

The default display and input format for any date is ***DD-MON-YY***. Valid Oracle dates are between January 1, 4712 B.C. and December 31, 9999 A.D.

Perhaps ***SYSDATE ()*** is most commonly used date function. It returns the current date and time of the database server. We can use this function just as any other column name or together with other Date functions to get some different result as and when required.

Example 3.4 below demonstrates the use of SYSDATE () function as any other column name and the output after executing it.

|  |  |  |
| --- | --- | --- |
| **Example 3.4: Using SYSDATE() Date Function**   |  | | --- | | **SELECT SYSDATE**  **FROM Dual;** | |  | |
|  |

**Arithmetic with Dates**

As we have mentioned earlier that database stores dates in numeric format i.e. as numbers so this automatically make it possible to perform calculations on the dates using arithmetic operators. We can add and subtract number constants as well as dates.

We can perform the following operations on dates:

|  |  |  |
| --- | --- | --- |
| **Operation** | **Result** | **Description** |
| Date + Number | Date | Adds a number of day(s) to a date. |
| Date – Number | Date | Subtract a number of day(s) from a date. |
| Date – Date | Number of Days | Subtract one date from another and returns the number of days between the two. |
| Date + Number/24 | Date | Adds a number of hours to a date. |

|  |  |  |
| --- | --- | --- |
| **Example 3.5: Using Arithmetic Operator with Dates**   |  | | --- | | **SELECT SYSDATE AS “Todays Date ”, (SYSDATE + 7) AS “Date after seven days”, (SYSDATE – (SYSDATE +7)) AS “Number of days between the two dates”**  **FROM Dual;** | |  | |
|  |

Example 31 above show how we can use an arithmetic operator with dates, first it show the todays date and then it adds number 7 to date to show us the date after seven days and lastly it shows us the number of days between the two dates. If a more current date is subtracted from an older date, the difference is a negative number.

Other than SYSDATE there are various date functions which are being used. The table below gives a complete description of all those functions.

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Purpose** | **Example** | **Result** |
| **ADD\_MONTHS**(*Date, n*) | This function returns the *date* plus *n* months. The value of *n* must be an integer and it can be negative. | **SELECT SYSDATE, ADD\_MONTHS(SYSDATE, 8) AS “After adding” FROM Dual;** | SYSDATE AFTER ADD  --------------- ----------------  04-AUG-11 04-APR-12 |
| **CURRENT\_DATE**() | This function doesn’t take any argument and returns the current date in the Georgian calendar for the session’s time zone. | **SELECT CURRENT\_DATE AS “Todays date” FROM Dual;** | TODAYS DA  ----------------  04-AUG-11 |
| **EXTRACT**(*c FROM date\_time | interval expression*) | This function extract and returns the specified component *c* of *date\_time | interval expression*. The valid components for this function are **YEAR, MONTH, DAY, HOUR, SECOND, TIMEZONE\_HOUR, TIMEZONE\_MINUTE, TIMEZONE\_REGION AND TIMEZONE\_ABBR** they must exist in an expression. | **SELECT SYSDATE AS “DATE”, EXTRACT( YEAR FROM SYSDATE) AS “YEAR”, EXTRACT(MONTH FORM SYSDATE) AS MONTH FROM Dual;** | DATE YEAR  -------------- ---------------  04-AUG-11 2011 |
| **LAST\_DAY**(*Date*) | This function returns the last day of the month for the specified *Date*. | **SELECT LAST\_DAY(SYSDATE) AS “Last Day” FROM Dual;** | LAST DAY  ------------------  31-AUG-11 |
| **MONTHS\_BETWEEN** (*Date1 , Date2*) | This function returns the number of months between *Date1* and *Date2* A whole number is returned if *Date1* and *Date2* are the same day of the month or if both are the last day of the month. The non-integer part of the result represents a portion of the month. | **SELECT MONTHS\_BETWEEN(’19-Dec-2008’,’31-Aug-2000’) AS months FROM Dual;** | MONTHS  ---------------  15.6129032 |
| **NEXT\_DAY**( *Date, Day\_of\_week*) | This function returns the date for the next day of the week followed by the *date*. | **SELECT NEXT\_DAY(’31-Dec-2011’, ‘Monday’) AS “1st Monday” FROM Dual;** | 1st MONDAY  -------------------  02-JAN-12 |
| **ROUND**(*Date [, Format]*) | This function returns *date* rounded to the unit specified by the *format*. If the *format* is omitted, date is rounded to the nearest day. | **SELECT ROUND(SYSDATE, ‘YEAR’) AS “Round Date” FROM Dual;** | ROUND DATE  ---------------------  01-JAN-12 |
| **TRUNC**(*Date [,Format]*) | This function returns *date* with time portion of the day truncated to the unit specified by the *format* | **SELECT TRUNC(last\_analyzed, ’HH’) FROM user\_tables WHERE table\_name=’BOOKS’;** | TRUNC(LAS  -----------------  25-JUL-11 |

The format models are covered later in this lesson.

**RR Date Format**

The RR date format is similar to YY element but this can be used to specify different centuries. This format can be used to vary the return value of the century as per the two-digit year value and the last two digits of the current year. The table below summarizes the behaviour of RR element.

|  |  |  |  |
| --- | --- | --- | --- |
|  | | If the specified two-digit year is : | |
| 0-49 | 50-99 |
| If two digits of the current years are: | 0-49 | The return date is in the current century | The return date is in the century before the current one |
| 50-99 | The return date is in the century after the current one | The return date is in the current century |

**Conversion Functions**

Conversion functions are used to convert a value from one datatype to another datatype. Generally the form of function name follows the convention *datatype To datatype.* The first datatype is the input datatype and the second one is the output datatype. This data type conversion can be done ***implicitly*** by Oracle server or ***explicitly*** by the user. Therefore we can say that a datatype conversion can be of two types:

**Data type Conversion**

**Explicit data type Conversion**

**Implicit data type Conversion**

Although the implicit data type conversion is available internally, it is highly recommended to do explicit data type conversion to ensure the reliability of SQL statement. Let’s discuss the two types of datatype conversions in detail.

**Implicit data type conversion:** These types of conversions are done by Oracle server whenever it supposed to be done. The implicit data type conversion is sub-divided into two categories:

1. For Assignment
2. For Expressions

For converting an ***assignment***, Oracle server automatically converts the data type of the value used in the assignment to that of the assignment target. The table below gives the details of the data type which automatically gets converted.

|  |  |
| --- | --- |
| **FROM** | **TO** |
| VARCHAR2 or CHAR | NUMBER |
| VARCHAR2 or CHAR | DATE |
| NUMBER | VARCHAR2 |
| DATE | VARCHAR2 |

For the expression evaluation the Oracle server uses one rule i.e. it converts those expression where conversion is needed and the rule for assignment conversions doesn’t covers them. The table below shows the data type which are automatically converted for the expression evaluation by the Oracle server.

|  |  |
| --- | --- |
| **FROM** | **TO** |
| VARCHAR2 or CHAR | NUMBER |
| VARCHAR2 or CHAR | DATE |

**Explicit data type conversion:** These types of conversion uses function to convert a value from one data type to another. As mentioned earlier it uses *data type TO data type* convention to convert a value. It basically uses TO\_<data type> to convert a value to the data type specified after the underscore. The picture below shows data type conversion from a data type to another.

**CHARACTER**

TO\_CHAR

TO\_CHAR

TO\_NUMBER

TO\_DATE

**NUMBER**

**DATE**

SQL provide three functions to convert a value from one data type to another. These functions are listed below; let’s discuss them in more detail.

**TO\_CHAR Explicit Data Type Conversion Function**

The Oracle TO\_CHAR function converts a number or date value to a VARCHAR2 character string. If the first data type is DATE type, it will be converted to a string representation. The resulting string style is specified with the datatype FORMAT elements which are optional parameters in the second parameter.

|  |
| --- |
| **Syntax:**  TO\_CHAR(number | date, [*format*] , [*nlsparams*]) |

The TO\_CHAR function can convert a DATE as well as NUMBER datatype; therefore for this reason the function follows some rules for them this is because the TO\_CHAR function is divided into two sub-categories for the two data types, these are:

* **TO\_CHAR Number Conversion:** The nlsparams parameter specifies the following characters, which are returned by number format elements:
  + - * Decimal Character
      * Group Separator
      * Local currency symbol
      * International currency symbol

If nlsparams or any other parameter is omitted, this function uses the default parameter values for the session.

|  |
| --- |
| **Syntax:**  TO\_CHAR(number, ‘format\_model’) |

There are few points to keep in mind when using TO\_CHAR function with number that, Oracle server rounds the stored decimal value to the number of decimal spaces provided in the format model.

Oracle server displays a string of hash (#) in place of a whole number whose digits exceed the number of digits provided in the format model.

Below is the list of the format elements that can be used with the TO\_CHAR function to display a number value as a character:

|  |  |
| --- | --- |
| **Element** | **Description** |
| **9** | The number of 9’s represent the number’s display width |
| **0** | Forces to display leading zeros |
| **$** | Places a floating dollar sign |
| **L** | Floating local currency symbol |
| **.** | Prints the decimal point in the position specified |
| **,** | Prints comma in position specified |
| **MI** | Minus signs to right of the number |
| **PR** | Parenthesize negative number |
| **EEE** | Scientific notations(format must specify four E’s) |
| **V** | Multiply by 10 n times (n= number of 9s after V) |
| **B** | Display zero values as blank, not 0 |

Example 3.6 demonstrates the use of TO\_CHAR function with numbers it takes the price of all the books in the Books table and Format it to be displayed with a Dollar ($) sign earlier to that the values in Price column were displayed without any sign.

|  |  |  |
| --- | --- | --- |
| **Example 3.6: Using TO\_CHAR with NUMBER data type**   |  | | --- | | **SELECT TO\_CHAR(Price,’$99,999.00’) AS PRICE**  **FROM Books;** | |  | |
|  |

* **TO\_CHAR Date Conversion:** the nlsparams parameter specifies the language in which month and day names and abbreviations are returned. If this parameter is omitted, this function uses the default data languages for the session.

|  |
| --- |
| **Syntax:**  TO\_CHAR(date, ‘format\_model’) |

The Oracle server displays the date value in DD-MON-YY format. The TO\_CHAR function can be used to convert a date from the default format to the one specified. But before doing so we must keep the following guidelines in mind regarding the format model, these are:

* It must be enclosed in single quotation mark and is case sensitive.
* It can include any valid date format element, be sure to separate the date value and format model by using a comma.
* The names of days and months in the output are automatically padded with blanks.
* Use fill mode *fm* element to remove any blank spaces or to supress any leading zeros.

The list of various date format elements for date is given below in two tables:

**Format Elements of Valid Date Formats**

|  |  |
| --- | --- |
| **Element** | **Description** |
| **SCC or CC** | Century; server prefixes B.B. dates with - |
| **Years in dates YYYY or SYYYY** | Year; server prefixes B.C. date with - |
| **YYY or YY or Y** | Last three, two or one digits of year |
| **Y,YYY** | Year with comma in this position |
| **IYYY, IYY, IY, I** | Four, three, two or one digit year based on the ISO standard |
| **SYEAR or YEAR** | Year spelled out; server prefixes B.C. with - |
| **BC or AD** | B.C. /A.D. indicator |
| **B.C. or A.D.** | B.C. /A.D. indicator with periods |
| **Q** | Quarter of year |
| **MM** | Month: two digit value |
| **MONTH** | Name of month padded with blanks to length of nine character |
| **MON** | Name of month, three letter abbreviation |
| **RM** | Roman numeral month |
| **WW or W** | Week of year or month |
| **DDD or DD or D** | Day of year, month or week |
| **DAY** | Name of day padded with blanks to a length of nine characters |
| **DY** | Name of day three letter abbreviation |
| **J** | Julian day; the number of days since 31 December 4713 B.C. |

We also need to format time as per our specifications; the table below gives the format element to display time information and literals and to change numerals to spelled numbers. Also the table of other Format is also given after that.

**Date Format Elements – Time Formats**

|  |  |
| --- | --- |
| **Element** | **Description** |
| **AM or PM** | Meridian indicator |
| **A.M. or P.M.** | Meridian indicator with periods |
| **HH or HH12 or HH24** | Hour of day, or hour(1-12), or hour(0-23) |
| **MI** | Minute(0-59) |
| **SS** | Second(0-59) |
| **SSSSS** | Seconds past midnight(0-86399) |

**Other Formats**

|  |  |
| --- | --- |
| **Element** | **Description** |
| **/ , .** | Punctuation is reproduced in the result |
| **“of the”** | Quoted string is reproduced in the result |

**Specifying Suffixes to influence Number Display**

|  |  |
| --- | --- |
| **Element** | **Description** |
| **TH** | Ordinal number for ex: DDTH for 4th |
| **SP** | Spelled out number for ex: DDSP for FOUR |
| **SPTH or THSP** | Spelled out ordinal numbers for ex: DDSPTH for FOURTH |

Our Database BOOKS does not contain date field in any of the table existing in the database. So to demonstrate the use of TO\_CHAR with dates we are making use of the DUAL table. The example 3.7 below demonstrates the use TO\_CHAR conversion function for the dates into a specified Date format. For any table column which store date in it we can simply replace the SYSDATE with name of the column or even we can hardcode the date value too.

|  |  |  |
| --- | --- | --- |
| **Example 3.7: Using TO\_CHAR with DATE data type**   |  | | --- | | **SELECT TO\_CHAR(SYSDATE, ‘fmDDTH Month YYYY’) AS “Formatted Date”**  **FROM Dual;** | |  | |
|  |

**TO\_NUMBER and TO\_DATE Explicit Data Type Conversion Functions**

Sometimes need arises to convert a character function either to a number or a date. We have seen the techniques in which we can convert a value to character using TO\_CHAR, in the same way we have two functions TO\_NUMBER and TO\_DATE conversion function to convert a character value to a number or date respectively. The syntax for the two functions is as follows:

|  |
| --- |
| **Syntax:**  TO\_DATE(char [, ‘format\_model’]) |
| **Syntax:**  TO\_NUMBER(char [, ‘format\_model’]) |

These functions have an *fx* modifier. This modifier specifies the exact matching for the character argument and date format model of a TO\_DATE function.

**General Functions**

The general functions are the functions which work with any data type and relates to the use of NULL values in the expression list. The functions which fall into this category are discussed in the table below:

|  |  |
| --- | --- |
| **Function** | **Purpose** |
| **NVL***(expr1,expr2)* | This function converts a null value to an actual value. This function can be used to convert any data type but the return value is always same as expr1. |
| **NVL***(expr1, expr2, expr3)* | This function examines *expr1* if it is not null then the function returns *expr2* else if it is null then the function returns *expr3.* |
| **COALESCE***(expr1, expr2, … expr n)* | This function returns the first non-null value in the list of values. This function can take multiple alternate values. |

## SOLVED PRACTICE QUESTIONS

**Practice Set – 3.1**

1. **Write a query to display the current date. Label the column as Date.**

**Solution:**

**SELECT SYSDATE AS “Date” FROM Dual;**

**Output:**

|  |
| --- |
|  |

1. **Which assertion about the following is most TRUE?**

**SELECT P\_Code || City FROM Publisher;**

**SELECT CONCAT (P\_Code, City) FROM Publisher;**

1. **If the P\_Code is NULL the first statement will not include that Publisher’s City.**
2. **If the P\_Code is NULL the second statement will not include that Publisher’s City.**
3. **Both statements will return the same data.**
4. **The second statement will raise an exception is the P\_Code is NULL for any Publisher.**

**Solution:**

**(C) Both the statements are equivalent.**

1. **Which function(s) accept argument of any datatype? (Choose all that apply.)**
2. **SUBSTR**
3. **NVL**
4. **ROUND**
5. **SIGN**
6. **COALESCE**

**Solution:**

**(B, E) because NVL and COALESCE are the general functions which accommodates the need of all or any datatype.**

1. **Write a query to display the title of the books in upper case and the length of the title. For all the books available whose price is less than £ 8?**

**Solution:**

**SELECT UPPER (Title) AS “Title”, Price, LENGTH (Title) AS “Length of Title”**

**FROM Books WHERE Price < 8;**

**Output:**

|  |
| --- |
|  |

1. **You want to display current date as the day, week, number and year. Which of the following will give output like the following?**

**Monday Week 35, 2011**

1. **SELECT to\_char(SYSDATE, ‘DOW Week WOY YYYY’) FROM Dual;**
2. **SELECT to\_char(SYSDATE, ’Day’ ||’ Week’ ||’ WOY, YYYY’) FROM Dual;**
3. **SELECT to\_char(SYSDATE, ‘Day “ Week” WW, YYYY’) FROM Dual;**
4. **SELECT to\_char(SYSDATE, ‘Day Week#, YYYY’) FROM Dual;**
5. **You cannot calculate week number with Oracle.**

**Solution:**

**(C) SELECT to\_char (SYSDATE, ‘Day “Week” WW, YYYY’) FROM Dual;**

1. **Which of the single row function is used to return a specific portion of a character string?**
2. **INSTR**
3. **SUBSTR**
4. **LPAD**
5. **LEAST**

**Solution:**

**(B) SUBSTR returns the portion of a character string.**

1. **Display the release dates of all the books releasing in month of May.**

**Solution:**

**SELECT B\_Code, Release\_Date**

**FROM Books**

**WHERE to\_Char(Release\_Date,’MON’) = ‘MAY’;**

**Output:**

|  |
| --- |
|  |

1. **Display the release date of all the books.**

**Solution:**

**SELECT B\_code, Release\_date FROM Written\_by;**

**Output:**

|  |
| --- |
|  |

1. **Display the B\_Code, Release\_Date and day of the week on which the book was released. Label the column as DAY.**

**Solution:**

**SELECT B\_Code, Release\_Date, to\_Char(Release\_Date,’DAY’) AS “ DAY “**

**FROM Written\_By;**

**Output:**

|  |
| --- |
|  |

## UNSOLVED PRACTICE QUESTIONS

**Note: For the unsolved practice question PRODUCT database is used which available in Annexure A.**

**Practice Set – 3.2**

1. Display the product name to be delivered with their delivery date and quantity to be delivered.
2. Display the name of the product, delivery date and day of the week on which the product is to be delivered by the supplier S3
3. Create a query to display the Product name, colour and price per quantity for all the products from PRODUCT table. Format the Price with a $ sign before the amount.
4. Write a query to display the following for each product to be delivered:

**Supplier <Supplier name> delivers <quantity> units of <product name> to the <department name> department. Label this as DELIVERY REPORT.**

1. Create a query to display the department and the total amount they need to pay at the time of delivery for each product. Format the price to be 10 characters long, left-padded with $.
2. Write a query to display all Product name and their colours whose name second character is ‘e’.
3. Write a query to display the names of all departments in caps.
4. Write a query to count the character in the departments name and label that column as Total Number of Characters.
5. Write a query to display the product name with their name in lower case and the length of the name.
6. Display the name of the product name with their first letter capitalized and all other letters lowercase and length of the names, for all the products whose name starts with B, L, or S. Give each column an appropriate label. Sort the result by the product names.

\*\*\* Chapter Ends \*\*\*