Single-Row Functions

Section Recap

This was a very important section!

Here you learned about SQL functions, which are those small programs that are built into the Oracle database and help you work with different types of data and perform some basic operations.

The functions covered in this section are usually called "single-row functions".

Most of these functions receive one or more expressions as parameters, and all of them return a single value for each row returned by the query.

We covered different types of functions:

First, we looked at TEXT FUNCTIONS, starting with the SUBSTR function, which allows you to extract a portion of a string.

This function takes 3 parameters. The first one is the original string, the second one is the position at which you want to start extracting text, and the third one, which is optional, defines the length of the substring you want to extract. If you omit this parameter, the extracted substring extends until the end of the original string.

We also looked at the UPPER, LOWER, and INITCAP functions, which are used to convert text to uppercase, lower case or just the initials of each word to uppercase, respectively.

You also learned that the REPLACE function can help you replace text within a string.

The first parameter you pass to this function is the original string. The second one is the text you want to replace, and the third one, which is optional, is the text you want to put in place of what you passed in the second parameter. If this parameter is omitted, the text passed in the second parameter is removed from the original string.

There's also a function to get the length of a string, which is called precisely LENGTH.

And finally, you learned that the INSTR function allows you to search for a string within another string, and returns the position where the string was found, or 0 if it wasn't.

This function takes 2 mandatory parameters, which are the original string, and the string you want to search for, but you can pass a third parameter to tell Oracle the position in which you want to start searching, and a fourth parameter, to specify the occurrence you are interested in.

Both the SUBSTR and INSTR functions can receive a negative number for the starting position parameter, which means that the search will start going backwards from the end of the original string.

Here is a query that demonstrates the text functions covered in the section:

```
SELECT SUBSTR ('SOME TEXT', 2, 3),
   SUBSTR ('SOME TEXT', 2),
   SUBSTR ('SOME TEXT', - 4, 3),
   UPPER ('hello'),
   LOWER ('HELLO'),
   INITCAP ('HELLO THERE'),
   REPLACE ('ONE DAY', 'DAY', 'YEAR'),
   REPLACE ('ONE DAY', 'DAY'),
   TRANSLATE ('HELLO THERE','ELH','314'),
   TRANSLATE ('HELLO THERE','ELH','314'),
   TRANSLATE ('HELLO THERE','LHE','143'),
   TRANSLATE ('HELLO THERE','*EL','*'),
   LENGTH ('LENGTH OF THIS TEXT'),
   INSTR ('THIS IS A STRING', 'I'),
   INSTR ('THIS IS A STRING', 'I',4),
   INSTR ('THIS IS A STRING', 'I',4,2),
   INSTR ('THIS IS A STRING', 'I',-1,3),
   CONCAT ('ONE ','TWO '),
   'ONE ' || 'TWO '
FROM DUAL;
```

In the numeric functions group, we looked at TRUNC and ROUND, which allow you to truncate or round a number to a determined number of decimals. If you pass a negative number for the second parameter, these functions truncate or round digits to the left of the decimal point.

In both functions the second parameter is optional, and when it is not passed, both functions return an integer.

We also looked at the FLOOR and CEIL functions, which work as if you rounded down or up, respectively, with no decimals, the number passed as parameter.

The SIGN function returns -1, 0, or 1 when the number passed as parameter is a negative number, zero or a positive number respectively.

And finally, we saw that the MOD function is used to get the remainder of one number divided by a second number.

Here is an example query of the numeric functions:

```
SELECT TRUNC (26.895),
   TRUNC (26.895,1),
   TRUNC (426.895,-2),
   ROUND (26.895),
   ROUND (26.895,1),
   ROUND (469.895,-2),
   FLOOR (26.895),
   CEIL (26.125),
   SIGN (-25),
   SIGN (0),
   SIGN (25),
   MOD (10,4)
FROM dual;
```

The next group we covered was DATE functions, and the first thing that was mentioned is that SYSDATE returns the date and time of the operating system where the database is running.

We saw also that ADD_MONTHS is used to add a certain number of months to a given date.

The original date is passed in the first parameter, and the number of months to add goes in the second parameter. If this number is negative, that number of months is subtracted from the original date.

Then we saw that LAST_DAY returns the date of the last day of the month that contains the date passed as a parameter, so if I pass SYSDATE, last_day would give me the date of the last day of the current month.

The MONTHS_BETWEEN function gives you the difference in months between the 2 dates passed as parameters. The return value of this function is a number, and it can contain decimals.

The NEXT_DAY function returns the date of the first weekday named by the second parameter, that is later than the date passed in the first parameter, so it can tell me, for example, when is the next Monday.

We saw that the TRUNC function can also be used with dates. This function returns the date passed in the first parameter truncated to the date or time portion defined in the second parameter, which is optional. If I omit the second parameter, this function removes the time component from the date, or in other words, it sets the time to midnight.

The ROUND function can also be used with dates, but it is not very frequently used that way.

A query showing the date functions covered is shown below.

```
SELECT
   SYSDATE,
   SYSTIMESTAMP,
   ADD_MONTHS (SYSDATE, 2),
   ADD_MONTHS (SYSDATE, - 2),
   LAST_DAY (SYSDATE),
   MONTHS_BETWEEN (SYSDATE, '22-dec-2018'),
   MONTHS_BETWEEN (SYSDATE, '22-sep-2019'),
   EXTRACT (YEAR FROM SYSDATE),
   EXTRACT (MONTH FROM SYSDATE),
   EXTRACT (HOUR FROM systimestamp),
   NEXT_DAY (SYSDATE, 'sunday'),
   NEXT_DAY (SYSDATE, 'sun'),
   TRUNC (SYSDATE),
   TRUNC (SYSDATE, 'hh'),
   TRUNC (SYSDATE, 'month'),
   TRUNC (SYSDATE, 'year'),
   ROUND (SYSDATE, 'year')
FROM dual;
```

And the last group we covered here was CONVERSION functions, which are used when you need to convert some data from one data type to another data type, for example, numbers to strings, strings to dates, etc.

Oracle can make some implicit conversions in several situations, but instead of relying on implicit conversions, it is better to make an explicit conversion, in which you decide how exactly the conversion is done.

In that lesson, we saw that the TO_CHAR function can be used to convert numbers into strings with a specified format, and a cheat sheet with the format models and modifiers was included in the lesson.

Remember that Oracle will add a leading blank space when you convert a positive number with TO_CHAR, but you can remove it by adding the FM format modifier, and also, that you can add an optional third parameter to specify some NLS settings, such as the currency symbol, group separator, decimal character, etc.

We also covered the opposite operation, which is converting strings into numbers with the TO_NUMBER function, and we saw that it uses the same format elements as the TO_CHAR function when used with numbers, and that it can also take the third argument for defining some NLS parameters.

Then we saw that the TO_CHAR function can also be used with dates, and it allows us to display a date in different formats, or to display only parts of it. There are a lot of format elements you can use with this function, which includes elements for the date and for the time component, and they were also included in the format models cheat sheet.

This function can also accept an argument for the NLS parameter to define the language to use for the month and day names.

And finally, we saw the TO_DATE function, which is used to convert strings into dates. This function takes the string you want to convert and optionally the format string and the NLS parameters argument.

Although the format string is optional, it is usually a bad idea not to use it, because in those cases Oracle would use the default date format of the current session, and thus, the same statement could produce different results when run on different systems.

This function is very important, because over the years I have seen that converting strings to dates is the most abused form of implicit type conversions.

I mentioned this more than once in some lectures at the beginning of the course, when I wrote some queries where I compared a date column to a string that looked like a date. That would work if I write the string in my session's date format, but the problem is that if the same statement is run on another system with different settings, the statement could fail. That's why at that point in the course I introduced the concept of "date literals".

If you want to use a string to compare it to a date, or to insert or update a date column, you MUST explicitly use the TO_DATE function and provide the appropriate format parameter if you want your statement to always run correctly regardless of the local settings of the system where it is being run.

The topic of implicit conversions is so important, that I added a whole lesson just to show and explain the dangers of using them.

You learned as well that the conversion functions allow you to define a default value to be returned when the conversion fails by means of the ON CONVERSION ERROR clause, and that there is a function specifically designed to check if a conversion is possible, which is called precisely VALIDATE_CONVERSION.

Later in the section, we looked at the DECODE function and the very important "CASE EXPRESSIONS", which are used to implement IF-THEN-ELSE or CASE logic in SQL statements. We saw that the DECODE function is an Oracle-specific function, and that it was the only way we had to implement this type of logic in older versions of the database.

A typical call to the DECODE function looks like this:

```
DECODE(department_id, 3, 'IT', 4, 'HR', 'OTHER')
```

This means: if department_id is equal to 3, then return 'IT'. If it is equal to 4 then return 'HR' and in any other case return 'OTHER'.

The decode function is very powerful, but we saw that CASE expressions are actually more flexible and readable, so, that is the recommended way to provide this type of logic if you are working with a database version that supports it, which is most likely the case.

You still need to understand how DECODE works, because you will most likely need to modify or maintain old code that uses it, so learn how to use it, but for new code prefer case expressions. We saw that there are 2 types of case expressions:

The first one is called "SIMPLE CASE expressions", and expressions of this type look like this:

```
CASE department_id
   WHEN 3 THEN 'IT'
   WHEN 4 THEN 'HR'
   ELSE 'OTHER'
END AS dept
```

Here, department_id is the value that will be compared to all of the values provided in the cases, using an equality comparison, so this is equivalent to the DECODE function shown before. And the second type is called "SEARCHED CASE expressions", and they look like this:

```
CASE
   WHEN department_id = 1 AND hire_date > DATE '2015-01-01' THEN 'New accountant'
   WHEN job_id = 'IT_PROG' AND salary < 2000 THEN 'Low pay programmer'
   ELSE 'Other'
END
```

This is the most powerful form of CASE expressions. Here you don't define a specific value to be compared, and you can have different types of complex conditions (not only equality comparisons) that can be independent of each other.

Also, remember that you can provide aliases to CASE expressions in the same way you would do it with table columns. The alias must be defined after the END keyword.

And at the end of the section, we talked about date arithmetic, and you learned that the basis for date arithmetic is that you can add and subtract numbers to and from dates, in which case Oracle interprets those numbers as numbers of days, so, SYSDATE + 1 would return tomorrow's date, and SYSDATE – 1 would give us yesterday's date. In consequence, you can also subtract one date from another date, and the results will be the difference in days.

Remember that you can also add or subtract fractions of days, not only whole days, so for example, this (SYSDATE + 6/24) would be six hours later than the current date and time.

I could have used 0.25 of a day instead, but it is usually more readable to write it as a fraction.

Here is a query that demonstrates the use of the conversion functions covered in this section:

```
SELECT '[' || TO_CHAR(1325, '9,999.999')|| ']',
   '[' || TO_CHAR (0.25, '00.99')|| ']',
   '[' || TO_CHAR (1325, '999')|| ']',
   '[' || TO_CHAR (- 1325, '9,999')|| ']',
   '[' || TO_CHAR (1325, 'fmL9,999', 'NLS_CURRENCY = "USD"')|| ']',
   '[' || TO_CHAR (1325, 'fmL9,999')|| ']',
   TO_NUMBER ('.05', '.99'),
   TO_NUMBER ('T5' DEFAULT -1 ON CONVERSION ERROR, '99'),
   TO_CHAR (sysdate, 'day'),
   TO_CHAR (sysdate, 'month'),
   TO_CHAR (sysdate, 'mm-dd-yyyy "TIME:" hh24:mi:ss'),
   TO_DATE ('2019150618', 'yyyyddmmhh24'),
   TO_DATE ('2019150618' DEFAULT NULL ON CONVERSION ERROR, 'ddmmyyyyhh24'),
   CAST ('mar15-2019 10:20:30' AS DATE, 'mondd-yyyy hh24:mi:ss'),
   CAST (1325 AS VARCHAR2(50)),
   CAST (sysdate AS VARCHAR2(20)),
   CAST ('$.05' AS NUMBER DEFAULT - 1 ON CONVERSION ERROR, '0.99'),
   VALIDATE_CONVERSION ('$.05' AS NUMBER, '0.99')
FROM dual;
```

And that was it!

You learned A LOT in this section, and you are ready to continue to the next one!

See you there!