

10

More Functions

In Chapter 6 you were introduced to a few numeric functions such as MAX(), AVG(), and MIN(). SQL provides many more functions to assist developers in retrieving data in a form most useful to end users. The functions discussed in this chapter will cover string, mathematical, conversion, and date functions.

CHAPTER OBJECTIVES

The objectives for this chapter are as follows:

- Understand the use of string functions for manipulation and format of character data.
- Understand the use of mathematical functions to manipulate numeric data.
- Understand the use of conversion functions to convert data types for processing.
- Understand the use of date functions for manipulation and format of dates.

String Functions

String functions are used to manipulate character data such as CHAR and VARCHAR. If you've worked with programming languages such as Visual Basic.NET and C++ some of the string functions, we will be discussing will be familiar to you. Some examples of character strings are as follows:

Example	Description
Thomas Smith	Person's name
287 Main St.	Street address
052-22-2311	Social security number
(840) 712-1234	Phone Number

Notice in the above table that both social security number and phone number are considered character data. The rule of thumb is that any number that CANNOT be used in calculations should be of character data type.

LEFT and RIGHT Functions

The LEFT functions and RIGHT functions are used to extract characters from strings of data starting from the left or right of the string. They require two parameters, also called arguments, to function. The first parameter is the actual string value while the second is the number of characters to extract. The syntax for these functions are as follows:

LEFT(the character string, integer representing number of characters from the left to extract)

RIGHT(the character string, integer representing number of characters from the right to extract)

Let's try the LEFT() function by extracting the first two characters in the words "Hello World" with the following SELECT statement. The result is shown in Figure 1.

```
SELECT LEFT('Hello World', 2)
```

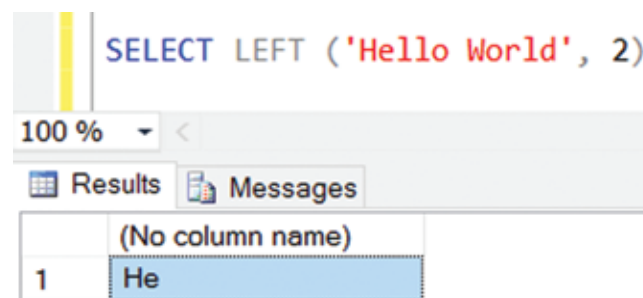


Figure 1

Notice the results contain the first two characters of “Hello World” starting from the left. Let’s use the same SELECT statement only in place of the LEFT() function we’ll use the RIGHT() function. The results are shown in Figure 2.

```
SELECT RIGHT('Hello World', 2)
```

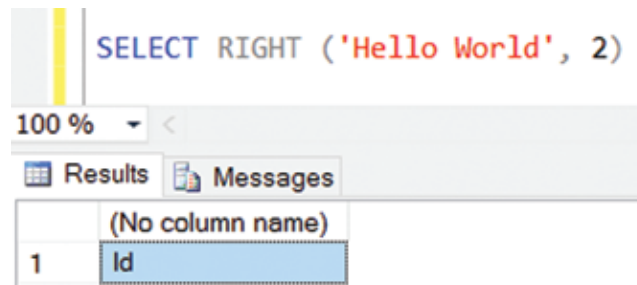


Figure 2

We can see by the results in Figure 2 that the RIGHT() function extracted the first two characters starting from the right. Next, let’s use the LEFT() function with a column from the employees data. Let’s select the first name as well as the first three characters of the first name using the SELECT statement below. The results are shown in Figure 3.

```
SELECT FirstName,
       LEFT(FirstName,3) 'First 3 Characters'
FROM Employees
```

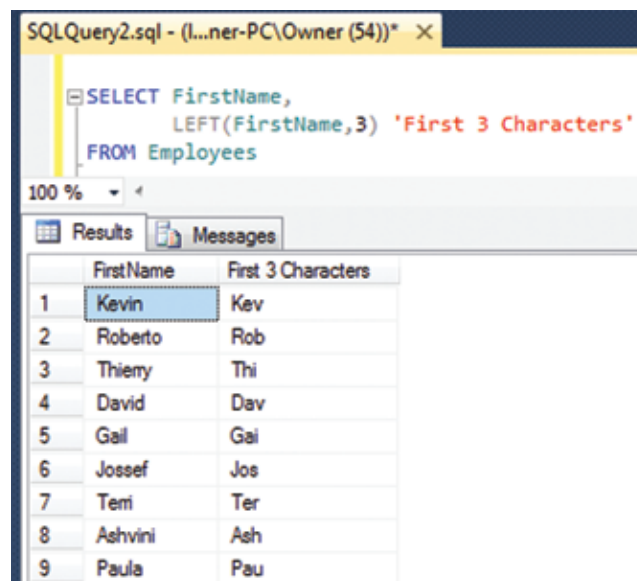


Figure 3

As you can easily see from Figure 3, the first three characters were extracted from the first name. You can also see that I provide an alias name for the function column called “First 3 Characters.”

LEN() Function

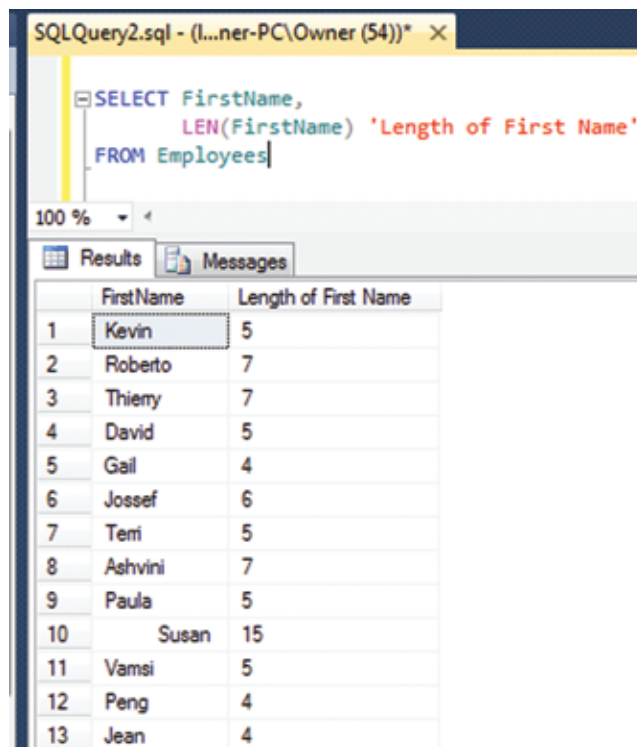
The LEN() function is useful when there is a need to find the length of a character string. It requires only one parameter. The syntax is shown below.

LEN(the character string)

Let's issue the following SELECT statement.

```
SELECT FirstName,
       LEN(FirstName) 'Length of First Name'
FROM Employees
```

The picture in Figure 4 shows part of the result set resulting from the above statement. You can see that the number of characters for each name is placed in the second column. Notice however, the name “Susan” and its length of 15. How can that be? This is because there are 10 spaces preceding the name “Susan” so that it is saved in the table as “Susan” rather than “Susan.”



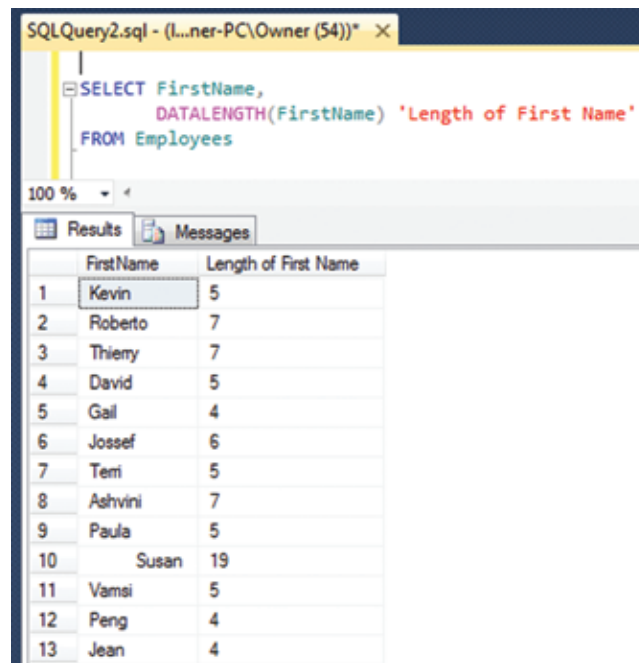
	FirstName	Length of First Name
1	Kevin	5
2	Roberto	7
3	Thierry	7
4	David	5
5	Gail	4
6	Josief	6
7	Teri	5
8	Ashvini	7
9	Paula	5
10	Susan	15
11	Vamsi	5
12	Peng	4
13	Jean	4

Figure 4

There are actually four trailing spaces on the name “Susan” as well but they are not being counted by the LEN() function. SQL Server has come up with another function to count both preceding and trailing spaces called DATALENGTH(). Let’s execute the same SELECT statement again but this time use DATALENGTH() in place of LEN() as follows:

```
SELECT FirstName,  
       DATALENGTH(FirstName) 'Length of First Name'  
FROM Employees
```

The result set produced by the above SELECT statement is shown in Figure 5 and you can see that the trailing spaces were counted this time. So why bother with the LEN() function if it doesn’t count the trailing spaces? Well, I decided to include it in this chapter because the DATALENGTH() function wasn’t available in earlier years so you might see the LEN() function in code when you are at your job.



	FirstName	Length of First Name
1	Kevin	5
2	Roberto	7
3	Thieny	7
4	David	5
5	Gail	4
6	Josief	6
7	Teri	5
8	Ashvini	7
9	Paula	5
10	Susan	19
11	Vamsi	5
12	Peng	4
13	Jean	4

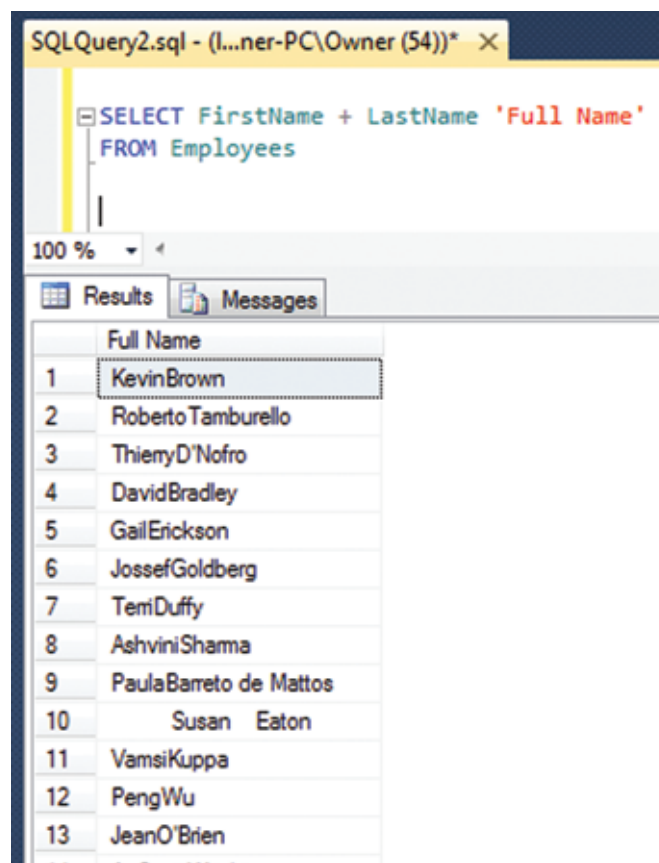
Figure 5

Concatenation

Before moving on to introducing more functions, I think it would be useful for you to understand concatenation of columns because the functions we are about to learn are very useful when using concatenation.

Let's say we need to produce a list of employees' first and last names in one column called "Full Name." We can concatenate the two columns using the plus sign as follows. The results are shown in Figure 6.

```
SELECT FirstName + LastName 'Full Name'  
FROM Employees
```



The screenshot shows a SQL query window titled "SQLQuery2.sql - (L...ner-PC\Owner (54))*". The query is: `SELECT FirstName + LastName 'Full Name' FROM Employees`. Below the query, the "Results" tab is active, showing a table with 13 rows. The first row is highlighted. The table has a single column labeled "Full Name".

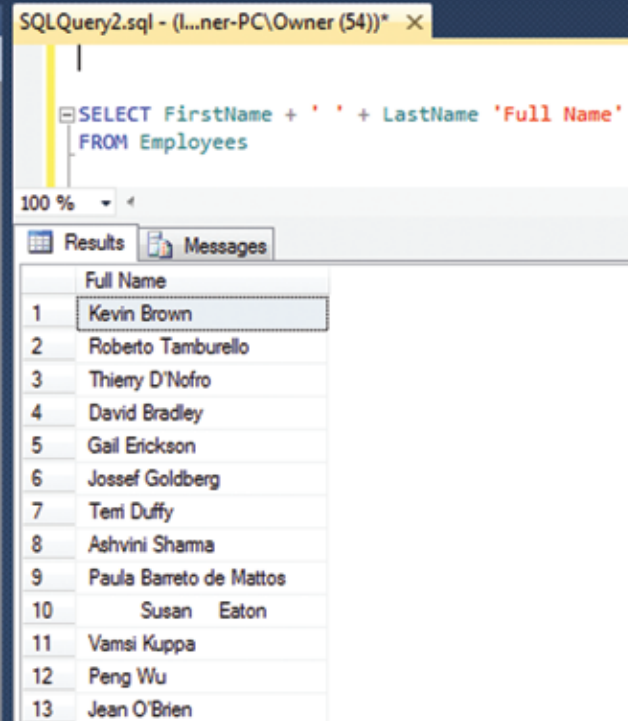
	Full Name
1	KevinBrown
2	RobertoTamburello
3	ThierryD'Nofro
4	DavidBradley
5	GailErickson
6	JossefGoldberg
7	TerriDuffy
8	AshviniSharma
9	PaulaBarreto de Mattos
10	Susan Eaton
11	VamsiKuppa
12	PengWu
13	JeanO'Brien

Figure 6

In Figure 6 notice that all the first and last names are concatenated, except for “Susan Eaton” which we will address later. As of now, there is no space between the two names for each employee. We need to concatenate a space between the two names as follows:

```
SELECT FirstName + ' ' + LastName 'Full Name'
FROM Employees
```

Execute the above SELECT statement and you should get the same results as shown in Figure 7. Notice that there is now a space between the first and last names.



	Full Name
1	Kevin Brown
2	Roberto Tamburello
3	Thierry D'Nofo
4	David Bradley
5	Gail Erickson
6	Josief Goldberg
7	Toni Duffy
8	Ashvini Shama
9	Paula Barreto de Mattos
10	Susan Eaton
11	Vamsi Kuppa
12	Peng Wu
13	Jean O'Brien

Figure 7

LTRIM and RTRIM Functions

The LTRIM() function is used to trim off spaces from the left of the string, whereas the RTRIM() function is used to trim off spaces from the right side of the string. Both accept only one parameter consisting of a string datatype. In our sample data, Susan Eaton has 10 spaces that precede her first name as well as 4 spaces that trail her first name. Let's work with Susan's name to understand the benefits of LTRIM() and RTRIM().

First, let's execute a SELECT statement on Susan's name without using either LTRIM() or RTRIM() as shown in Figure 8. The result shows preceding and trailing spaces on the first name.

```
SELECT FirstName + ' ' + LastName 'Full Name'
FROM Employees
WHERE EmployeeID = 34
```

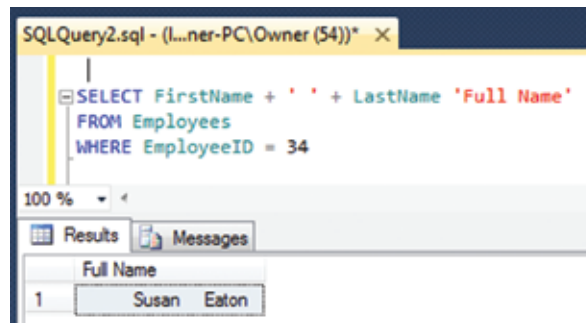


Figure 8

Next, let's use the LTRIM() function on Susan's first name. The results are shown in Figure 9. You can see that the preceding spaces are gone but the trailing spaces on Susan's first name remain.

```
SELECT LTRIM(FirstName) + ' ' + LastName 'Full Name'
FROM Employees
WHERE EmployeeID = 34
```

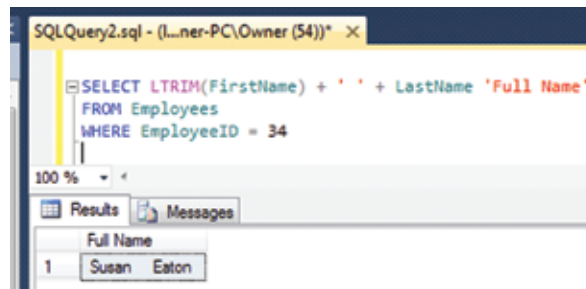


Figure 9

Let's now use the RTRIM() function to remove trailing spaces on Susan's first name as shown in Figure 10. You can see that the trailing spaces are now gone but the preceding spaces are back.

```
SELECT RTRIM(FirstName) + ' ' + LastName 'Full Name'
FROM Employees
WHERE EmployeeID = 34
```

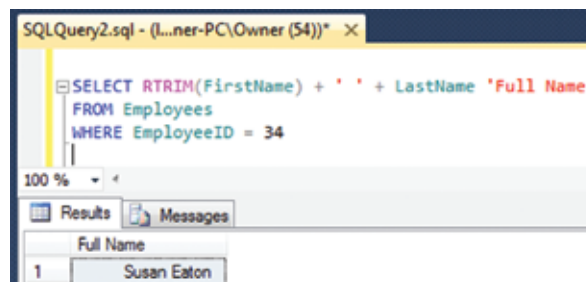


Figure 10

To remove both preceding and trailing spaces at the same time we need to use a function within a function. Figure 11 shows an example of this. First the RTRIM() function is used with the FirstName as the parameter. The result of that is the first name without trailing spaces. That result is then used as a parameter for the LTRIM() function where the preceding spaces are removed.

```
SELECT LTRIM(RTRIM(FirstName)) + ' ' + LastName 'Full Name'
FROM Employees
WHERE EmployeeID = 34
```

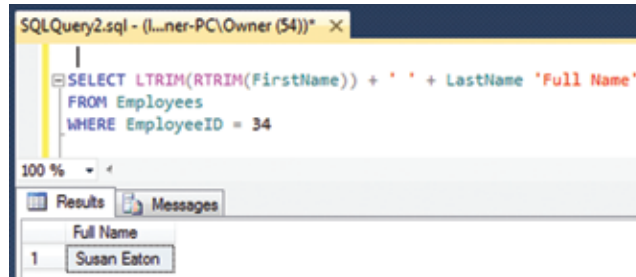


Figure 11

UPPER and LOWER Functions

The UPPER() function is used to convert character string data to uppercase while the LOWER() function is used to convert character string to lowercase. Let's try this out on the first name of employees by using a SELECT statement to retrieve employees' first names in its original form, first name in uppercase and first name in lower case. Figure 12 shows the SELECT statement to perform this along with the results.

The screenshot shows a SQL query window titled 'SQLQuery2.sql - (L...ner-PC\Owner (54))'. The query is:

SELECT FirstName,

UPPER(FirstName) Uppercase,

LOWER(FirstName) Lowercase

FROM Employees

The 'Results' tab is active, showing a table with 14 rows and 3 columns: First Name, Uppercase, and Lowercase.

	First Name	Uppercase	Lowercase
1	Kevin	KEVIN	kevin
2	Roberto	ROBERTO	roberto
3	Thierry	THIERRY	thierry
4	David	DAVID	david
5	Gail	GAIL	gail
6	Jossef	JOSSEF	jossef
7	Terri	TERRI	terri
8	Ashvini	ASHVINI	ashvini
9	Paula	PAULA	paula
10	Susan	SUSAN	susan
11	Vamsi	VAMSI	vamsi
12	Peng	PENG	peng
13	Jean	JEAN	jean
14	A. Scott	A. SCOTT	a. scott

Figure 12

CHARINDEX() Function

The CHARINDEX Function is used to find the starting position of a substring within a string value. For example, the starting position of “p” in the string “chapter” is 4 as shown in Figure 13. It isn’t case sensitive as Figure 14 shows us the same example but with an uppercase “P.”

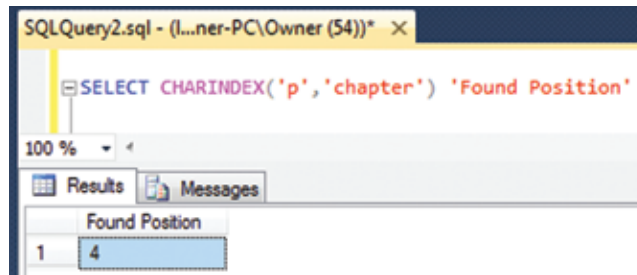


Figure 13

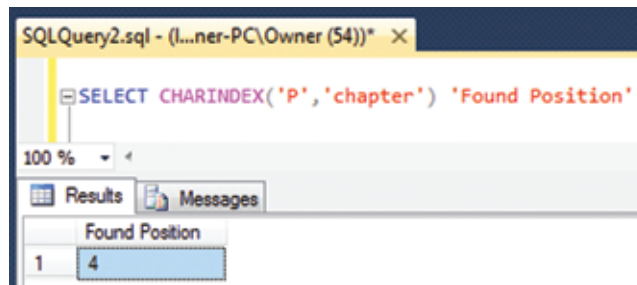


Figure 14

In the above examples there are two parameters associated with the CHARINDEX() function. There is an option to add one more parameter to represent the starting position in which to start searching for the substring. For example, in the character string “Mississippi” we can instruct the CHARINDEX() function to start searching for “s” at the 5th character as shown in Figure 15. Notice result set retrieved an answer of 6. That’s because all the “s” characters were ignored until after the 4th position.

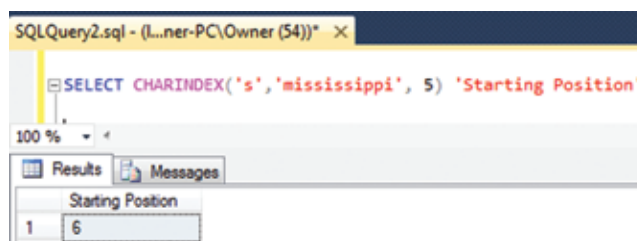


Figure 15

We are also able to search for more than one character. For example, to find the starting position of the word “walk” in the sentence “Let us walk by the sea” you could execute the statement shown in Figure 16.

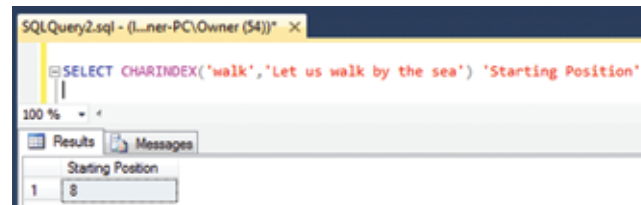


Figure 16

The CHARINDEX() function can be very helpful when trying to identify rows with certain substrings. For example, what if the manager would like to find all addresses that contain the characters “Dr.” in it. Any address that contains “Dr.” would return an integer value greater than zero, whereas addresses that don’t contain the characters would return zero. See the example in Figure 17 to demonstrate this.

The screenshot shows a SQL Query window titled 'SQLQuery2.sql - (L...ner-PC\Owner (54))'. The query text is: `SELECT AddressLine1, CHARINDEX('Dr.',AddressLine1) 'Starting Position' FROM Addresses`. Below the query, the 'Results' tab is active, displaying a table with two columns: 'AddressLine1' and 'Starting Position'. The table contains 18 rows of data, with the 'Starting Position' column showing 0 for most rows and 14 for '9297 Kenston Dr.' and 18 for '8036 Summit View Dr.'.

	AddressLine1	Starting Position
40	3397 Rancho View Drive	0
41	2137 Birchwood Dr	0
42	3928 San Francisco	0
43	475 Santa Maria	0
44	1185 Dallas Drive	0
45	7883 Missing Canyon Court	0
46	5452 Corte Gilberto	0
47	3238 Laguna Circle	0
48	7640 First Ave.	0
49	2294 West 39th St.	0
50	9297 Kenston Dr.	14
51	9687 Shakespeare Drive	0
52	1397 Paradise Ct.	0
53	3030 Blackburn Ct.	0
54	9605 Pheasant Circle	0
55	2425 Notre Dame Ave	0
56	8036 Summit View Dr.	18
57	213 Stonewood Drive	0

Figure 17

If we look at the subset of results in Figure 17, we can see that there are three blue highlighted rows but only two rows that contain the characters “Dr.” and the starting position is greater than zero. The first address, “2137 Birchwood Dr” does not contain the period so that is why the starting position amounts to zero.

If we wanted to just list those addresses that contain the characters “Dr.” we could use the CHARINDEX() function in the WHERE clause instead as shown in Figure 18.

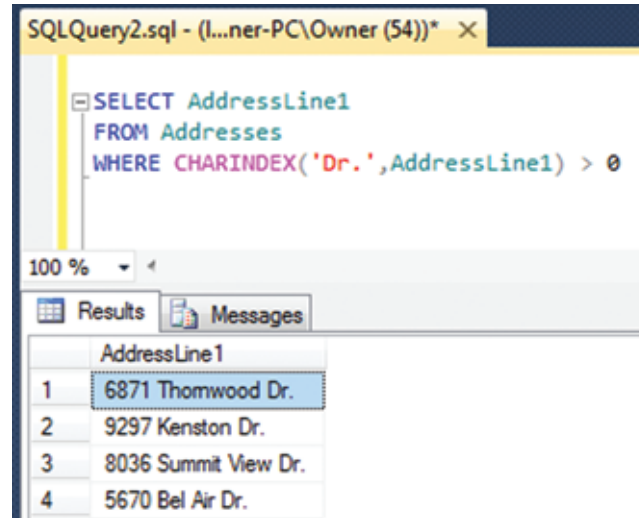


Figure 18

REPLACE() Function

The REPLACE() function is used to scan a character string and replace all references to a character string to another character string. It accepts three parameters: the character string, the substring to be replaced, and the substring that will replace it. See the syntax below:

REPLACE(the character string, character substring to be replaced, character substring to replace with)

Figure 19 shows an example of replacing “St.” with “Street.”

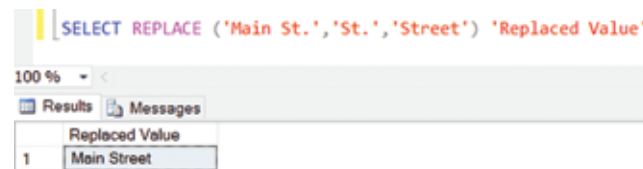
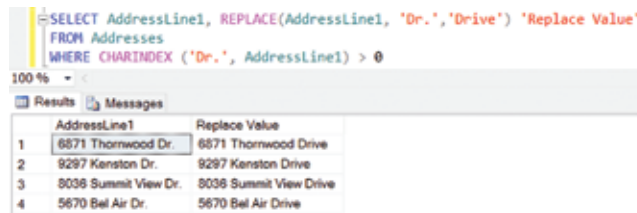


Figure 19

To go a step further, let's use the earlier example in Figure 18 to replace “Dr.” with “Drive.” Figure 20 shows the SQL statement along with the results. I included the original AddressLine1 in the first column of the result set so you can see the change.



```

SELECT AddressLine1, REPLACE(AddressLine1, 'Dr.', 'Drive') 'Replace Value'
FROM Addresses
WHERE CHARINDEX ('Dr.', AddressLine1) > 0

```

	AddressLine1	Replace Value
1	6671 Thornwood Dr.	6671 Thornwood Drive
2	9297 Kenston Dr.	9297 Kenston Drive
3	8038 Summit View Dr.	8038 Summit View Drive
4	5670 Bel Air Dr.	5670 Bel Air Drive

Figure 20

SUBSTRING() Function

The SUBSTRING() function is used to extract a number of characters starting with a given position and ending with the total number of characters desired to extract. The SUBSTRING() function accepts three parameters as shown in the example syntax below.

SUBSTRNG(the character string, starting position, number of characters to extract)

Figure 21 shows an example of extracting a subset of a character string starting in the 10th position and extracting six characters resulting with the value “Design.”



```

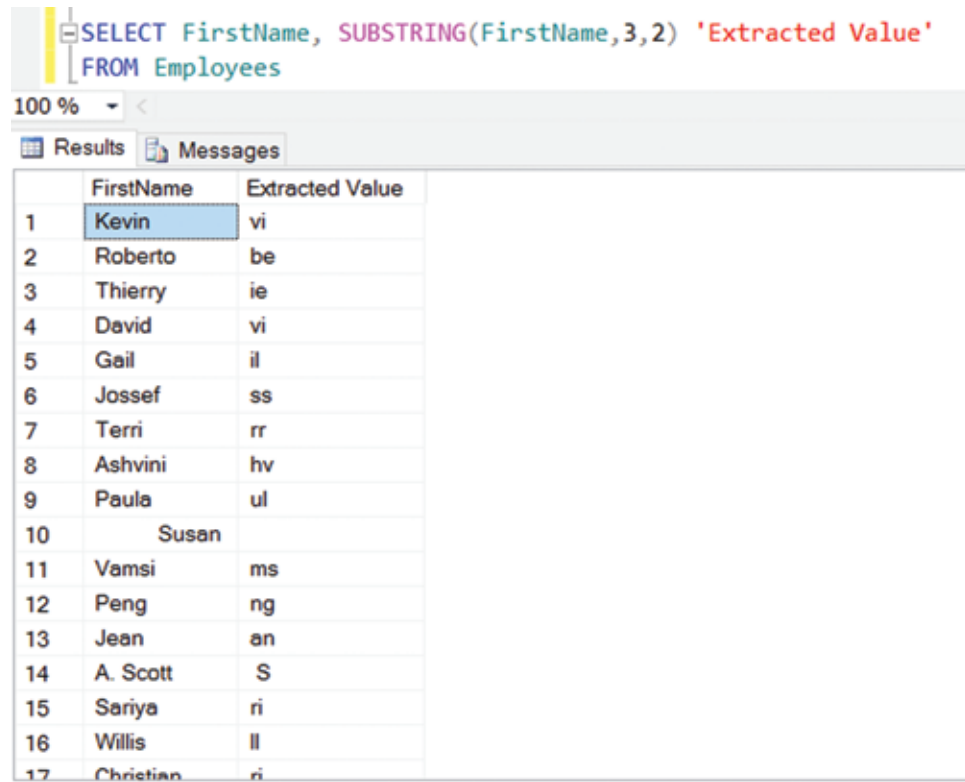
SELECT SUBSTRING('Database Design Class', 10, 6) 'Extracted Value'

```

	Extracted Value
1	Design

Figure 21

Figure 22 provides another example of using the SUBSTRING() function using the FirstName from the Employees table. The first column contains the full first name, while the second column contains the extracted value.



```
SELECT FirstName, SUBSTRING(FirstName,3,2) 'Extracted Value'
FROM Employees
```

	FirstName	Extracted Value
1	Kevin	vi
2	Roberto	be
3	Thierry	ie
4	David	vi
5	Gail	il
6	Jossef	ss
7	Terri	rr
8	Ashvini	hv
9	Paula	ul
10	Susan	
11	Vamsi	ms
12	Peng	ng
13	Jean	an
14	A. Scott	S
15	Sariya	ri
16	Willis	ll
17	Christian	ri

Figure 22

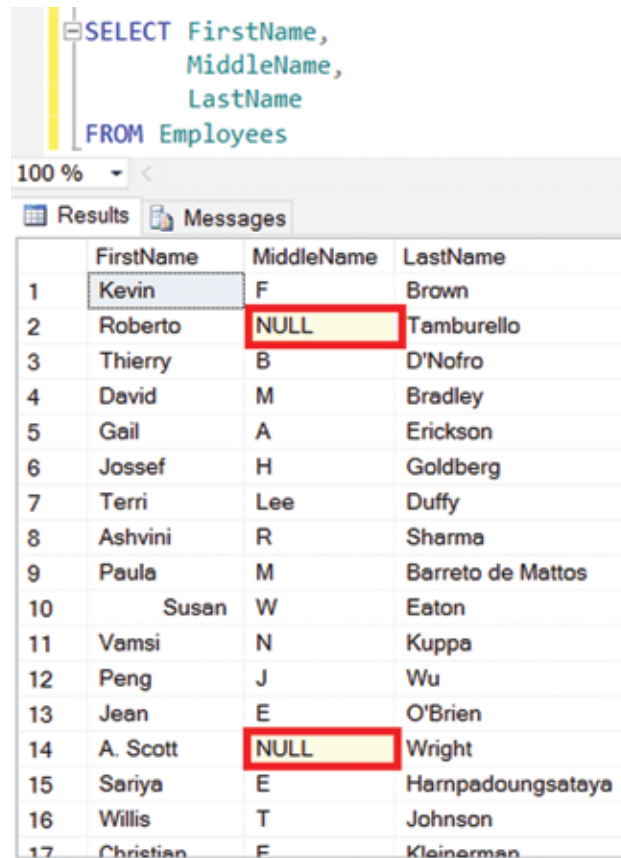
ISNULL() Function

When columns that contain NULL values are displayed they are displayed with the word

“NULL.” Often users request that something be substituted for the word “NULL.” The ISNULL() function can be used for this purpose. The ISNULL() function takes two parameters: one for the value that will be checked whether or not it is null and the other is the substitute value if it is null. The syntax for the ISNULL() function is as follows:

ISNULL (value to be determined, value of substitution)

In the example shown in Figure 23 we can see that two rows contain null values for middle name. They are pointed out with red arrows.

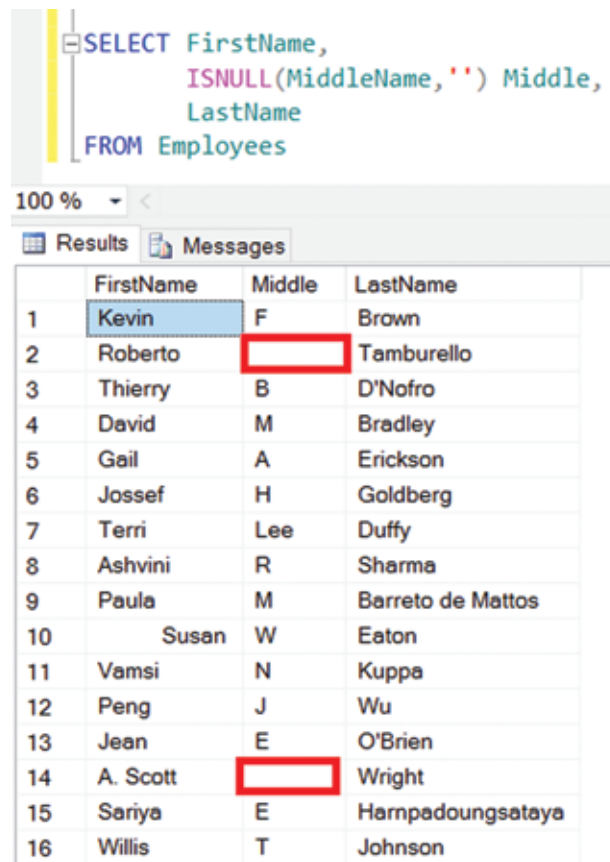


```
SELECT FirstName,
MiddleName,
LastName
FROM Employees
```

	FirstName	MiddleName	LastName
1	Kevin	F	Brown
2	Roberto	NULL	Tamburello
3	Thierry	B	D'Nofro
4	David	M	Bradley
5	Gail	A	Erickson
6	Jossef	H	Goldberg
7	Terri	Lee	Duffy
8	Ashvini	R	Sharma
9	Paula	M	Barreto de Mattos
10	Susan	W	Eaton
11	Vamsi	N	Kuppa
12	Peng	J	Wu
13	Jean	E	O'Brien
14	A. Scott	NULL	Wright
15	Sariya	E	Harnpadoungsataya
16	Willis	T	Johnson
17	Christian	F	Kleinerman

Figure 23

The example in Figure 24 executes the same SQL statement but uses the ISNULL() function to replace null values with a blank space. The red arrows point out where the blank spaces were substituted for the null values.



```

SELECT FirstName,
       ISNULL(MiddleName, ' ') Middle,
       LastName
FROM Employees

```

	FirstName	Middle	LastName
1	Kevin	F	Brown
2	Roberto		Tamburello
3	Thierry	B	D'Nofro
4	David	M	Bradley
5	Gail	A	Erickson
6	Jossef	H	Goldberg
7	Terri	Lee	Duffy
8	Ashvini	R	Sharma
9	Paula	M	Barreto de Mattos
10	Susan	W	Eaton
11	Vamsi	N	Kuppa
12	Peng	J	Wu
13	Jean	E	O'Brien
14	A. Scott		Wright
15	Sariya	E	Harnpadoungsataya
16	Willis	T	Johnson

Figure 24

MATH Functions

There are many mathematical functions available with SQL Server and the SQL language. In this section, we will be covering some of the more significant functions used for manipulating numeric data.

ABS() Function

The ABS() function is used to retrieve the absolute value of a numeric expression. For example, the SELECT statement in Figure 25 returns a value of 5.

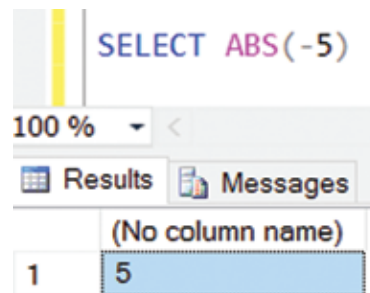


Figure 25

Figure 26 shows an example of using the Employees table to retrieve employees' vacation hours, subtracting vacation hours from 40 and, lastly, retrieving the absolute value of subtracting vacation hours from 40.

The screenshot shows a SQL query editor with the following text: `SELECT VacationHours, 40-VacationHours 'Resulting Value', ABS(40-VacationHours) 'Absolute Value' FROM Employees`. Below the editor, there are tabs for 'Results' and 'Messages'. The 'Results' tab is active, showing a table with four columns: 'VacationHours', 'Resulting Value', and 'Absolute Value'. The table contains 16 rows of data.

	VacationHours	Resulting Value	Absolute Value
1	42	-2	2
2	2	38	38
3	9	31	31
4	40	0	0
5	5	35	35
6	6	34	34
7	1	39	39
8	70	-30	30
9	54	-14	14
10	99	-59	59
11	95	-55	55
12	81	-41	41
13	65	-25	25
14	44	-4	4
15	45	-5	5
16	99	-59	59

Figure 26

POWER() Function

The POWER() function is used to raise a number to a given power. The POWER() function requires two parameters. The number to be raised and the power number. Figure 27 shows an example of using the POWER() function.

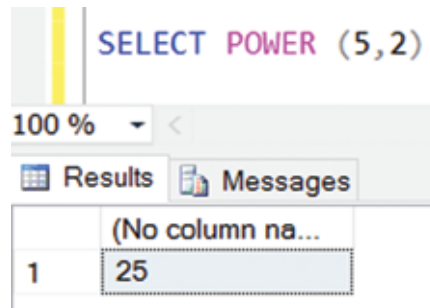


Figure 27

SQRT() Function

The SQRT() function provides the square root value of a number. Figure 28 demonstrates this.

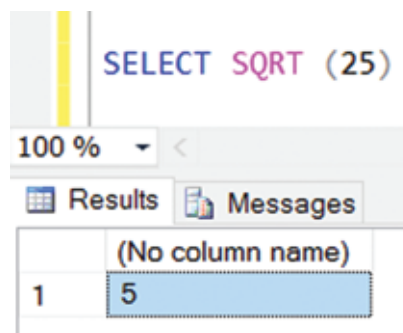


Figure 28

SQUARE() Function

The SQUARE() function is used to retrieve the square of a number. Figure 29 shows an example of using the SQUARE function.

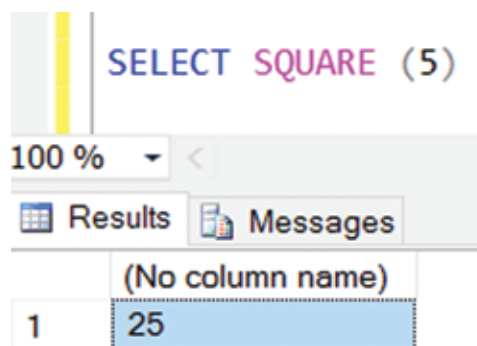


Figure 29

ROUND() Function

The ROUND() function is a very valuable function because when numeric data is retrieved it may need to be rounded to a given set of decimal numbers. It takes two parameters: the value to be rounded and the number of decimal places to be round to. This is shown in Figure 30. The value 123.4567 is rounded up to two decimal places. However, the original numbers are replaced with zeros. Not what we want. Not a problem. The next section discusses the CONVERT() function, which will provide the functionality we need for proper rounding.

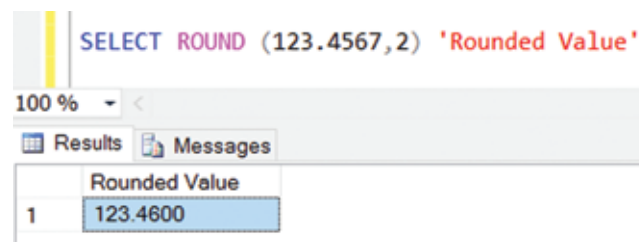


Figure 30

CONVERT() Function

The CONVERT() function is a very useful in converting data to facilitate manipulation and formatting. The CONVERT() function requires at least two parameters with a third parameter being optional and used for style. This section will show a few helpful ways to use the CONVERT() function.

Let's start with the example we used earlier with the ROUND() function. Although this function was successful in rounding, it left trailing zeros which we don't want. We are looking to only have two decimal numbers. To achieve this we are going to use the CONVERT() function to convert numeric values to decimals with two decimal places. The CONVERT() function uses two parameters: the first being the data type to be converted to and the second being the value that is to be converted. The syntax is shown below.

CONVERT (data type, value to convert)

Figure 31 shows an example of using the CONVERT() function along with the results from the conversion. The decimal data type parameter in the CONVERSION() function requires numeric parameters. The first being the size of the numeric value including decimals, the second being the number of decimal places.

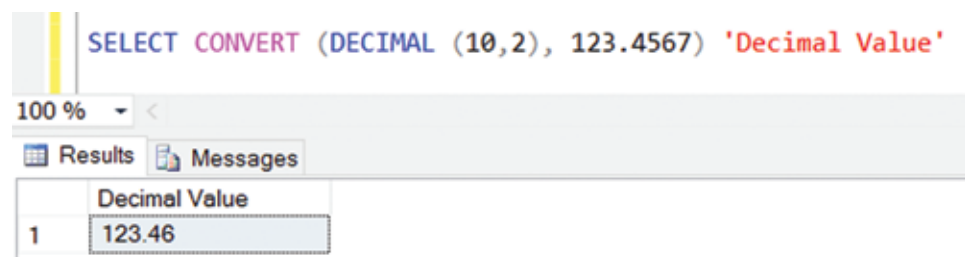


Figure 31

Notice that the value of 123.4567 was rounded up and only two decimal places exist. Unlike the `ROUND()` function, the trailing zeros are gone.

The example in Figure 32 uses three decimal places as another example of using the `CONVERT()` function. Notice how the third decimal in the original value is rounded up to 7.



Figure 32

The example in Figure 33 uses five decimal places. Since there are only four decimals in the original number, a zero is concatenated to the decimals to equal five places.

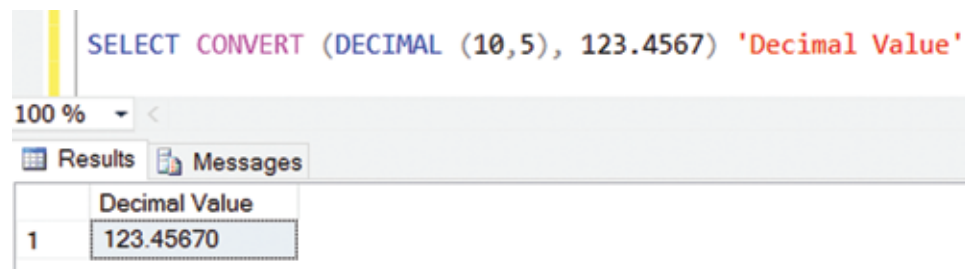


Figure 33

So far, we've used a total size of 10 digits in our conversion examples. What happens if we use a number not large enough? Figure 34 shows an example of the error that will occur when the size of the numeric value is not large enough to represent the value to be converted. In this case the size should be at least six since the value has three digits to the left of the decimal and the conversion is asking for three digits to the right of the decimal.

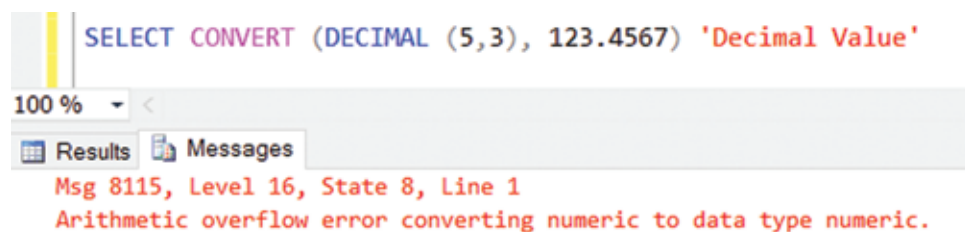


Figure 34

Figure 35 shows an example where the total size has been increased to six which is just right for the number we are trying to convert.

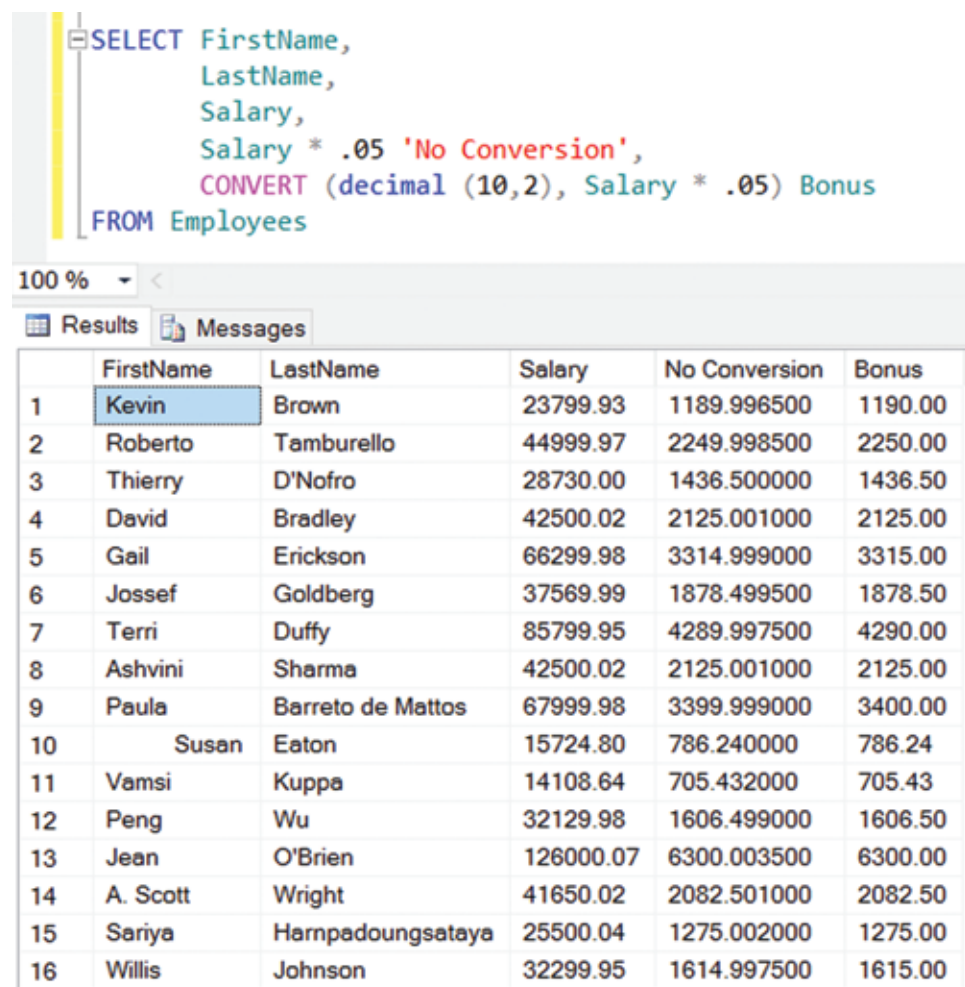


The screenshot shows a SQL query in the query editor: `SELECT CONVERT (DECIMAL (6,3), 123.4567) 'Decimal Value'`. Below the query, the 'Results' tab is active, displaying a single row with the column 'Decimal Value' and the value '123.457'.

	Decimal Value
1	123.457

Figure 35

Let's use the CONVERT() function on a query using our Employees table. Let's say the manager would like to see a list of employees, their salaries, and a bonus equal to 5% of their salaries. The query and results are shown in Figure 36.



The screenshot shows a SQL query in the query editor: `SELECT FirstName, LastName, Salary, Salary * .05 'No Conversion', CONVERT (decimal (10,2), Salary * .05) Bonus FROM Employees`. Below the query, the 'Results' tab is active, displaying a table with 6 columns: FirstName, LastName, Salary, No Conversion, and Bonus. The table contains 16 rows of employee data.

	FirstName	LastName	Salary	No Conversion	Bonus
1	Kevin	Brown	23799.93	1189.996500	1190.00
2	Roberto	Tamburello	44999.97	2249.998500	2250.00
3	Thierry	D'Nofro	28730.00	1436.500000	1436.50
4	David	Bradley	42500.02	2125.001000	2125.00
5	Gail	Erickson	66299.98	3314.999000	3315.00
6	Jossef	Goldberg	37569.99	1878.499500	1878.50
7	Terri	Duffy	85799.95	4289.997500	4290.00
8	Ashvini	Sharma	42500.02	2125.001000	2125.00
9	Paula	Barreto de Mattos	67999.98	3399.999000	3400.00
10	Susan	Eaton	15724.80	786.240000	786.24
11	Vamsi	Kuppa	14108.64	705.432000	705.43
12	Peng	Wu	32129.98	1606.499000	1606.50
13	Jean	O'Brien	126000.07	6300.003500	6300.00
14	A. Scott	Wright	41650.02	2082.501000	2082.50
15	Sariya	Hampadounsataya	25500.04	1275.002000	1275.00
16	Willis	Johnson	32299.95	1614.997500	1615.00

Figure 36

For your understanding, I included a column to show the 5% calculation without a decimal conversion and then next to it the Bonus column composed of the calculation and conversion.

As you can see, the `CONVERSION()` function is very useful in formatting numeric data. It can also be used to convert string data to numeric for calculations. For example, if we try to add the two strings together shown in Figure 37 the result will be a concatenation.

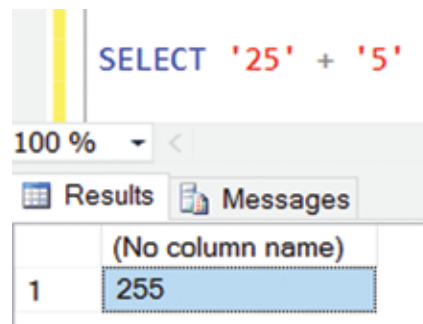


Figure 37

However, if we convert the strings to integer first as shown in Figure 38 addition will be performed.



Figure 38

CAST() Function

The `CAST()` function can also be used to convert data. It requires two parameters and the keyword "AS." Below is the syntax.

CAST (value to be converted **AS** data type to be converted to)

To show how the `CAST()` function works let's use the example from Figure 38 but substitute the `CONVERT()` function for the `CAST()` function. Figure 39 shows an example of using the `CAST()` function.



Figure 39

While both CAST() and CONVERT() functions can be used to convert data to other data types, there are some differences between the two as listed below.

- The CONVERT() function is specific to SQL Server while the CAST() function is an ANSI standard and can be used by other databases.
- While CONVERT() can be used for formatting purposes CAST() cannot.
- CONVERT() is especially useful for formatting dates while CAST() is not.

Date Functions

GETDATE() Function

The GETDATE() function is used to retrieve the current date and time from the computer's operating system. You can use the GETDATE() function in a SELECT statement as shown in Figure 40.

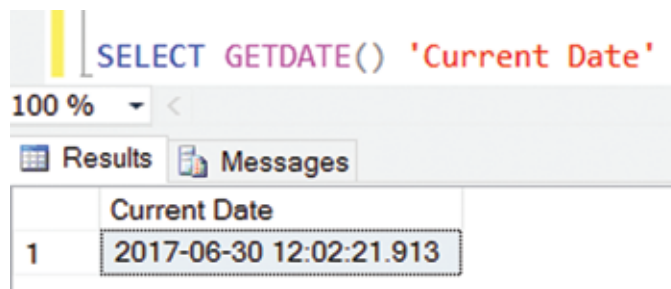


Figure 40

The date in Figure 40 represents the date and time I executed the statement on Jun 30, 2017 at 12:02 (the example shows military time). The format of the date shown is the default. Users don't usually find this format user friendly and request a more readable format. Luckily, we are able to use the CONVERT() function to present a number of various formats.

To format dates, we'll need an additional parameter to represent the style of the date. The syntax is as follows:

CONVERT (character data type and size, the date, the style)

Various numbers are used to represent styles for dates. The example in Figure 41 shows the current date in a MM/DD/YYYY format.

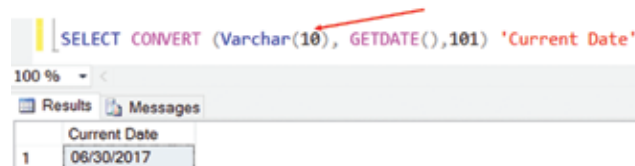


Figure 41

Notice the Varchar() size of 10. This is the size of the resulting date. Figure 42 is another example of using the CONVERT(). This time the date will be formatted as MMM DD, YYYY.

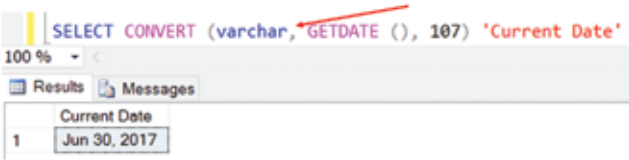


Figure 42

Notice data type wasn't provided in the example used in Figure 42. This is ok as SQL Server will still present the format correctly without providing the size. However, if you do provide a size and it's not large enough the result will be truncated as shown in Figure 43.

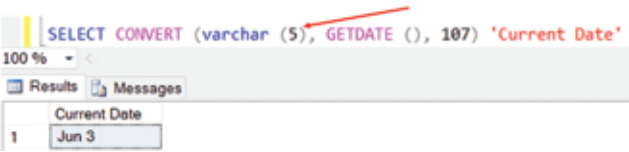


Figure 43

The table below shows some of the most common date formats that are used in the United States, along with their SQL syntax and example.

Format	SQL	Example
Mon DD YYYY	<code>SELECT CONVERT(VARCHAR(20), GETDATE(), 100)</code>	Jun 30, 2017
HH:MM(AM)or PM		
MM/DD/YY	<code>SELECT CONVERT(VARCHAR(8), GETDATE(), 1)</code>	06/30/17
MM/DD/YYYY	<code>SELECT CONVERT(VARCHAR(10), GETDATE(), 101)</code>	06/30/2017
MM-DD-YY	<code>SELECT CONVERT(VARCHAR(8), GETDATE(), 10)</code>	06-30-17
MM-DD-YYYY	<code>SELECT CONVERT(VARCHAR(10), GETDATE(), 110)</code>	06-30-2017

DATEPART() Function

The DATEPART() function is used to return any part of the date, month, day, year. It accepts one parameter which represents the part of the date you would like to retrieve. Figure 44 is an example of using the DATEPART() function to retrieve month, day, and year from the employees' birth dates.

```
SELECT FirstName,
       LastName,
       BirthDate,
       DATEPART(MONTH,BirthDate) Month,
       DATEPART(Day,BirthDate) Day,
       DATEPART(YEAR,BirthDate) Year
FROM Employees
```

	FirstName	LastName	BirthDate	Month	Day	Year
1	Kevin	Brown	1977-06-03	6	3	1977
2	Roberto	Tamburello	1964-12-13	12	13	1964
3	Thierry	D'Nofro	1949-08-29	8	29	1949
4	David	Bradley	1965-04-19	4	19	1965
5	Gail	Erickson	1942-10-29	10	29	1942
6	Jossef	Goldberg	1949-04-11	4	11	1949
7	Terri	Duffy	1961-09-01	9	1	1961
8	Ashvini	Sharma	1967-04-28	4	28	1967
9	Paula	Barreto de Mattos	1966-03-14	3	14	1966
10	Susan	Eaton	1968-03-20	3	20	1968
11	Vamsi	Kuppa	1967-04-19	4	19	1967
12	Peng	Wu	1966-04-19	4	19	1966
13	Jean	O'Brien	1966-01-13	1	13	1966
14	A. Scott	Wright	1958-10-19	10	19	1958
15	Sariya	Harnpadoungsa...	1977-06-21	6	21	1977
16	Willis	Johnson	1968-08-18	8	18	1968
17	Christian	Kleinerman	1966-02-18	2	18	1966

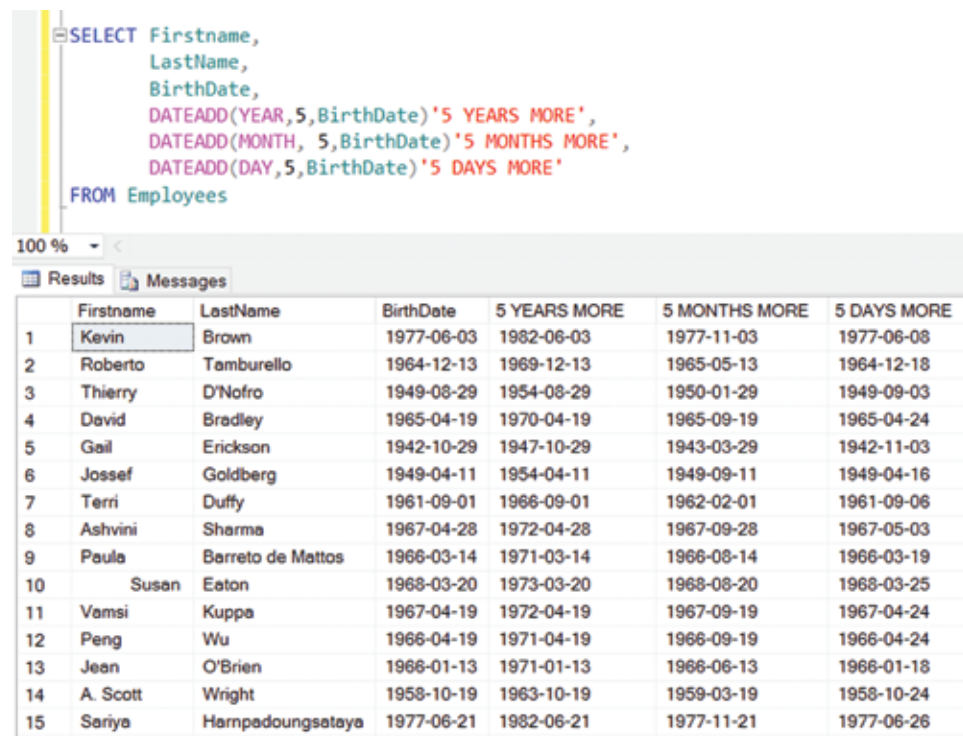
Figure 44

DATEADD() Function

The DATEADD() function is useful for adding years, months, or days to a date. It takes three parameters as shown in the syntax below.

DATEADD (date part, number to add by, date value)

Figure 45 is an example of using the DATEADD() function to retrieve information showing the results of adding 5 years, 5 months, and 5 days to the employees' birth dates.



```

SELECT Firstname,
       LastName,
       BirthDate,
       DATEADD(YEAR,5,BirthDate)'5 YEARS MORE',
       DATEADD(MONTH, 5,BirthDate)'5 MONTHS MORE',
       DATEADD(DAY,5,BirthDate)'5 DAYS MORE'
FROM Employees

```

	Firstname	LastName	BirthDate	5 YEARS MORE	5 MONTHS MORE	5 DAYS MORE
1	Kevin	Brown	1977-06-03	1982-06-03	1977-11-03	1977-06-08
2	Roberto	Tamburello	1964-12-13	1969-12-13	1965-05-13	1964-12-18
3	Thierry	D'Nofro	1949-08-29	1954-08-29	1950-01-29	1949-09-03
4	David	Bradley	1965-04-19	1970-04-19	1965-09-19	1965-04-24
5	Gail	Erickson	1942-10-29	1947-10-29	1943-03-29	1942-11-03
6	Jossef	Goldberg	1949-04-11	1954-04-11	1949-09-11	1949-04-16
7	Terri	Duffy	1961-09-01	1966-09-01	1962-02-01	1961-09-06
8	Ashvini	Sharma	1967-04-28	1972-04-28	1967-09-28	1967-05-03
9	Paula	Barreto de Mattos	1966-03-14	1971-03-14	1966-08-14	1966-03-19
10	Susan	Eaton	1968-03-20	1973-03-20	1968-08-20	1968-03-25
11	Vamsi	Kuppa	1967-04-19	1972-04-19	1967-09-19	1967-04-24
12	Peng	Wu	1966-04-19	1971-04-19	1966-09-19	1966-04-24
13	Jean	O'Brien	1966-01-13	1971-01-13	1966-06-13	1966-01-18
14	A. Scott	Wright	1958-10-19	1963-10-19	1959-03-19	1958-10-24
15	Sariya	Harnpadoungsataya	1977-06-21	1982-06-21	1977-11-21	1977-06-26

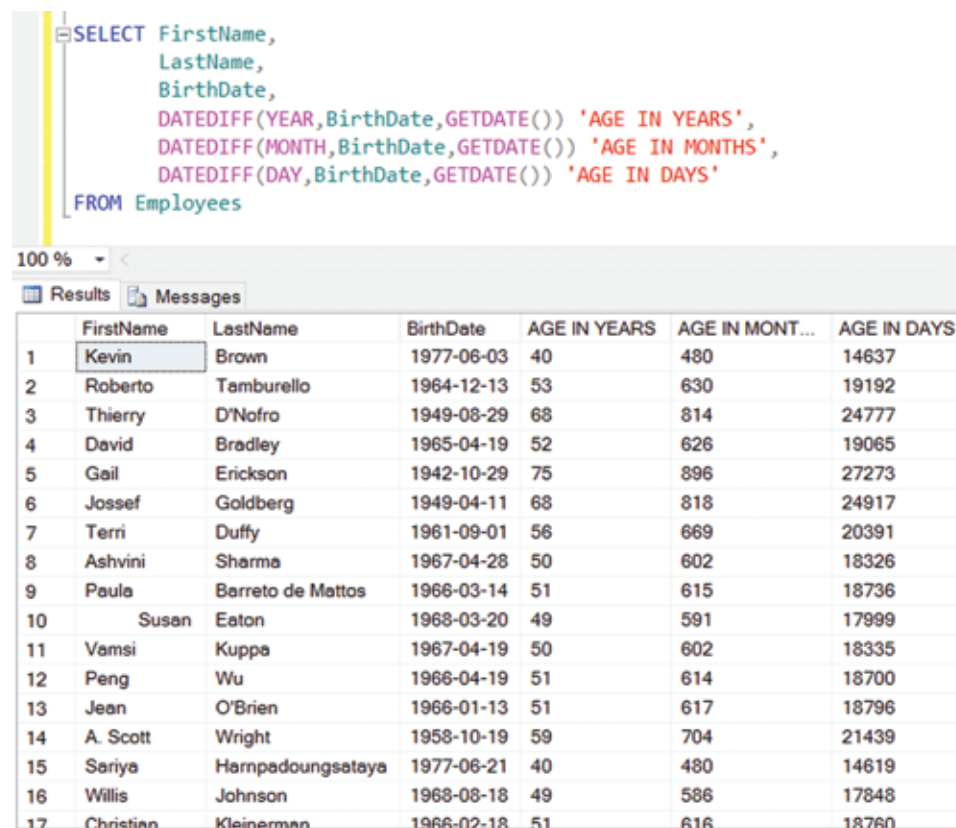
Figure 45

DATEDIFF() Function

The DATEDIFF() function is used to show the difference between two dates. It takes three parameters as shown in the syntax below:

DATEDIFF (date part, start date, end date)

Figure 46 shows an example of using the DATEDIFF() function to show the difference between the employees' birth dates and the current date in years, months, and days. As you can see this is an efficient method to retrieve an employee's age.



```

SELECT FirstName,
       LastName,
       BirthDate,
       DATEDIFF(YEAR,BirthDate,GETDATE()) 'AGE IN YEARS',
       DATEDIFF(MONTH,BirthDate,GETDATE()) 'AGE IN MONTHS',
       DATEDIFF(DAY,BirthDate,GETDATE()) 'AGE IN DAYS'
FROM Employees

```

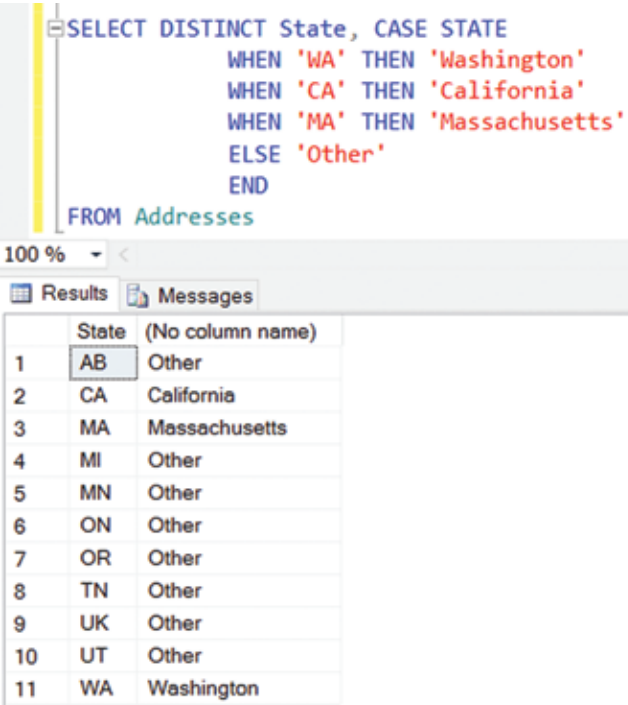
	FirstName	LastName	BirthDate	AGE IN YEARS	AGE IN MONTHS	AGE IN DAYS
1	Kevin	Brown	1977-06-03	40	480	14637
2	Roberto	Tamburello	1964-12-13	53	630	19192
3	Thierry	D'Nofro	1949-08-29	68	814	24777
4	David	Bradley	1965-04-19	52	626	19065
5	Gail	Erickson	1942-10-29	75	896	27273
6	Josief	Goldberg	1949-04-11	68	818	24917
7	Terri	Duffy	1961-09-01	56	669	20391
8	Ashvini	Sharma	1967-04-28	50	602	18326
9	Paula	Barreto de Mattos	1966-03-14	51	615	18736
10	Susan	Eaton	1968-03-20	49	591	17999
11	Vamsi	Kuppa	1967-04-19	50	602	18335
12	Peng	Wu	1966-04-19	51	614	18700
13	Jean	O'Brien	1966-01-13	51	617	18796
14	A. Scott	Wright	1958-10-19	59	704	21439
15	Sariya	Harnpedoungsataya	1977-06-21	40	480	14619
16	Willis	Johnson	1968-08-18	49	586	17848
17	Christian	Kleinerman	1966-02-18	51	616	18760

Figure 46

CASE Statement

Although CASE is called a statement and not a function, I felt that it qualifies to be in this chapter because, like functions, it is a useful feature. The CASE statement can be used when the output is dependent on the value from a column in the database.

As an example, Figure 47 uses the CASE statement to write out the state name based on the initials from the database. Notice the “ELSE” that outputs the word “Other” if neither WA, CA, nor MA is present.



```
SELECT DISTINCT State, CASE STATE
                        WHEN 'WA' THEN 'Washington'
                        WHEN 'CA' THEN 'California'
                        WHEN 'MA' THEN 'Massachusetts'
                        ELSE 'Other'
                        END
FROM Addresses
```

100 %

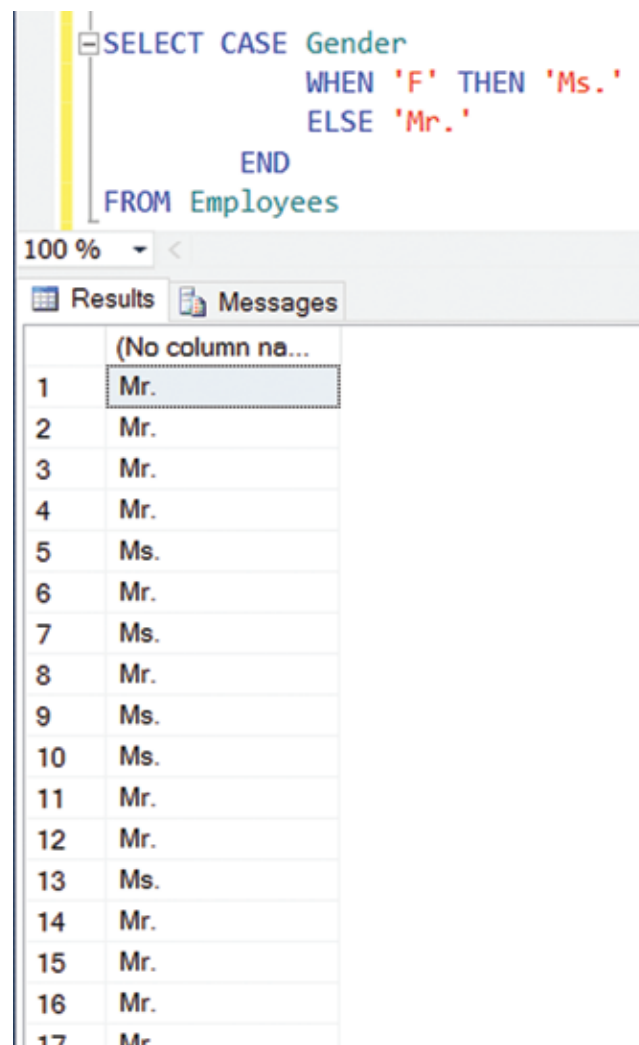
Results Messages

	State	(No column name)
1	AB	Other
2	CA	California
3	MA	Massachusetts
4	MI	Other
5	MN	Other
6	ON	Other
7	OR	Other
8	TN	Other
9	UK	Other
10	UT	Other
11	WA	Washington

Figure 47

As another example, let's say the manager would like to see a list of formal names such as "Mr. John L Smith." We would need to use the CASE statement in two sections. One, to determine gender and two, to determine whether a middle name exists.

Let's take it step by step and determine gender first. Figure 48 shows us how to do that.



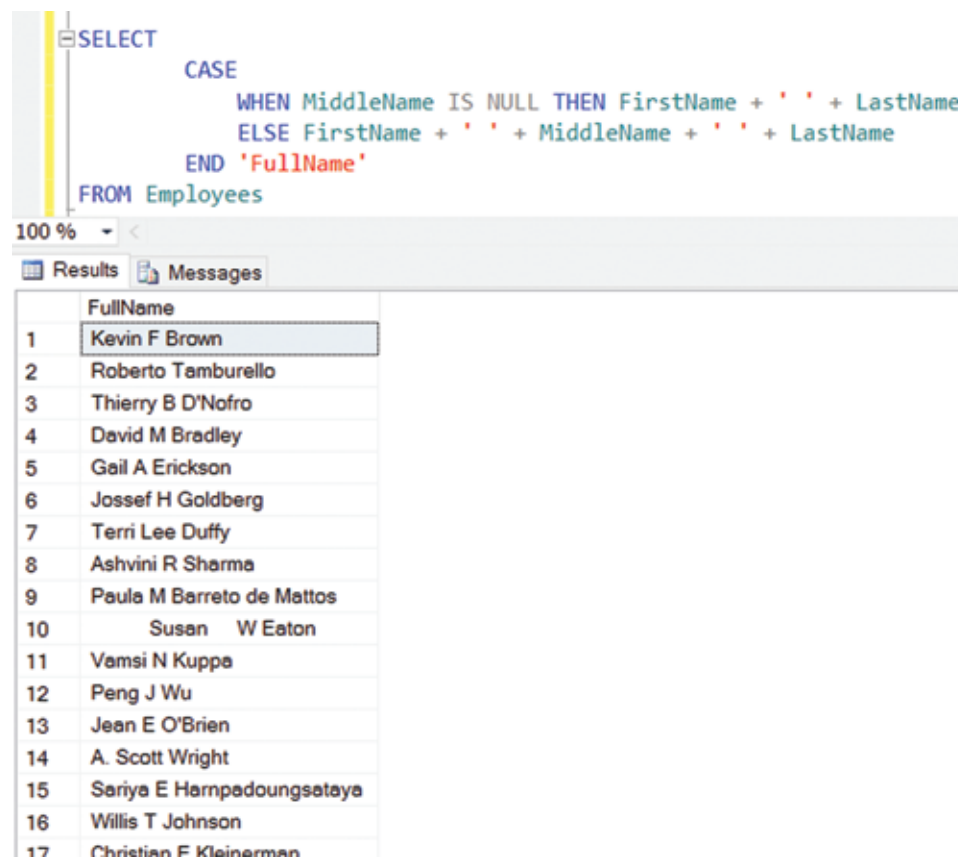
```
SELECT CASE Gender
        WHEN 'F' THEN 'Ms.'
        ELSE 'Mr.'
      END
FROM Employees
```

	(No column na...)
1	Mr.
2	Mr.
3	Mr.
4	Mr.
5	Ms.
6	Mr.
7	Ms.
8	Mr.
9	Ms.
10	Ms.
11	Mr.
12	Mr.
13	Ms.
14	Mr.
15	Mr.
16	Mr.
17	Mr.

Figure 48

As you can see from Figure 48 when the Gender equal “F” then “Ms.” is returned, otherwise “Mr.” is returned. Notice how there is a space after both “Ms.” and “Mr.” This is because it will be concatenated to the employee name. Next, we need to concatenate the employee’s name to each surname. Before we do that let’s look at the employee name portion in Figure 48.

The CASE statement is being used to determine whether the middle name is NULL. If it is null then only the first name, a space and the last name need to be concatenated. If it is NOT NULL then the first name, a space, middle name, another space, and last name need to be concatenated. I provided an alias of “FullName” to the result.



```

SELECT
    CASE
        WHEN MiddleName IS NULL THEN FirstName + ' ' + LastName
        ELSE FirstName + ' ' + MiddleName + ' ' + LastName
    END 'FullName'
FROM Employees

```

100 %

Results Messages

	FullName
1	Kevin F Brown
2	Roberto Tamburello
3	Thierry B D'Nofro
4	David M Bradley
5	Gail A Erickson
6	Jossef H Goldberg
7	Terri Lee Duffy
8	Ashvini R Sharma
9	Paula M Barreto de Mattos
10	Susan W Eaton
11	Vamsi N Kuppa
12	Peng J Wu
13	Jean E O'Brien
14	A. Scott Wright
15	Sariya E Hampedoungsetaya
16	Willis T Johnson
17	Christian F Kleinerman

Figure 49

Now we need to combine the two CASE statements with a plus sign to concatenate them so that you achieve the final results shown in Figure 49.

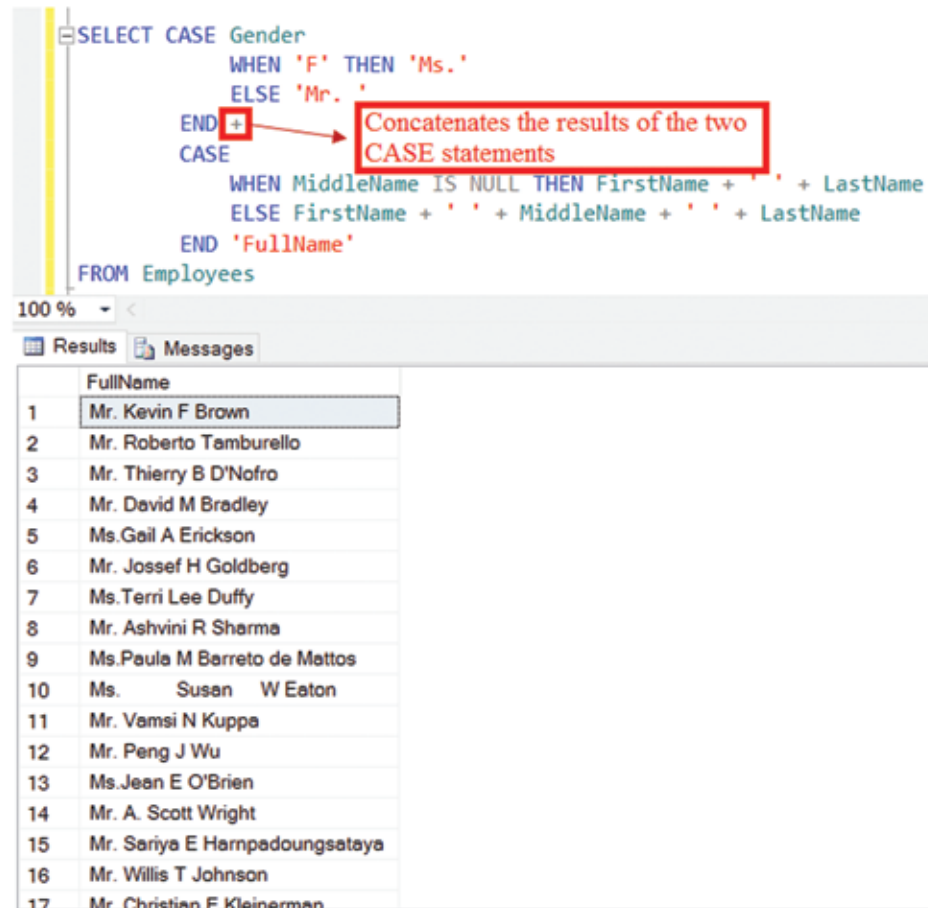


Figure 49

Summary

This chapter demonstrated how to use many of SQL Server's functions to manipulate and format data for a useful display of information. This chapter also covered the use of string functions to manipulate character data such as CHAR and VARCHAR. Furthermore, this chapter introduced you to many of the mathematical functions in SQL Server that are used to manipulate numeric data, along with conversion functions to convert data types for processing. Date functions for manipulation and formatting were also explained and a few of the most common date formats were covered along with their syntax. Lastly, the CASE statement was explained and you were shown the advantage of concatenating CASE statements to create useful results.

Exercises

1. Retrieve the employees' first names, the first three characters of the employees' first names, and the length of the employees' first names.
2. Concatenate the employees' first, middle, and last names. Use the ISNULL() function to replace NULL values with a blank space.
3. Retrieve all uppercase characters for the first names of the employees and all lowercase for the last names of employees using the UPPER and LOWER functions.
4. Using the CHARINDEX() function, list the first names and last names of employees whose first names contain the characters "er".
5. Retrieve AddressLine1 from the Addresses table and replace "Dr." and "Dr" with "Drive". (Hint: Use the CHARINDEX function)
6. Retrieve three characters from the first name of employees, starting with the second character.
7. For each employee, retrieve the first name, last name, and salary amount and show a bonus of 15% with two decimal places.
8. Retrieve the first name, last name, and birth date of all employees. Show the birth date in MM/DD/YYYY format.
9. Retrieve the first name, last name, hire date, and in individual columns, the month, day, and year of hire date for each employee.
10. For each Design Engineer in the Employees table, retrieve the first name, last name, and salary amount, and show a bonus of 5% with two decimal places.
11. Using the ABS() function, retrieve the resulting and absolute value of all employees with over 20 sick leave hours subtracted from 20. Order by sick leave hours.
12. Use the CONVERT() function along with the GETDATE() function to convert the current date to varchar and use the MM/DD/YYYY format.
13. Using the DATEDIFF() function, display how many years each employee has been working since their hire date.
14. Use the CASE operator to assign the following ranks to each distinct Group Name in the Departments table.
 - Executive General and Administration = "Top Management"
 - Research and Development = "Specialists"
 - Manufacturing = "Laborers"
 - ELSE = "Middle Management"
 - Label the column "Category"
15. Use the CASE operator to assign the following categories to State in the Addresses table.
 - CA = "Canada"
 - UK = "United Kingdom"
 - ELSE = "United States"
 - Call the column name = "Country"
 - Order by "Country"